*NASA Conference Publication 10012*
*Part 1*

# NASA Workshop on Computational Structural Mechanics 1987

Nancy P. Sykes, *Editor*
*Analytical Services and Materials, Inc.*
*Hampton, Virginia*

Proceedings of a workshop sponsored by
the National Aeronautics and Space
Administration and held at
Langley Research Center
Hampton, Virginia
November 18–20, 1987

February 1989

(NASA-CP-10012-Pt-1) NASA WORKSHOP ON COMPUTATIONAL STRUCTURAL MECHANICS 1987, PART 1 (NASA. Langley Research Center) 383 p CSCL 20K
H1/39
N89-29773
--THRU--
N89-29788
Unclas
0211941

National Aeronautics and
Space Administration

# Preface

This document contains the proceedings of the NASA Workshop on Computational Structural Mechanics, held at NASA Langley Research Center, November 18-20, 1987. The workshop was sponsored jointly by NASA Langley Research Center and NASA Lewis Research Center.

The purpose of the workshop was to allow participants in Langley's and Lewis' Computational Structural Mechanics (CSM) research programs to meet and to share research objectives and accomplishments. The intent was to encourage a cooperative Langley/Lewis CSM program in which Lewis concentrates on engine structures applications, Langley concentrates on airframe and space structures applications, and all participants share technology of mutual interest.

The workshop was organized into the following three sessions:

I Concurrent Processing Methods and Applications

II Advanced Methods & Testbed/Simulator Development

III Computational Dynamics

Session I dealt with parallel processing methods and languages, new computer hardware, and software architecture to exploit parallel computers.

Session II dealt with the Langley CSM Testbed, the Lewis Engine Structures Computational Simulator, and Structural Analysis Technology involving finite elements, boundary elements, and probabilistic approaches.

Session III dealt with advanced methods for structural dynamics.

The use of trade names or names of manufacturers in this publication does not constitute an official endorsement of such products or manufacturers, either expressed or implied, by the National Aeronautics and Space Administration.

W. Jefferson Stroud

**With the exception of a few adjustments made primarily for the purpose of uniformity, all papers have been published as received.**

**—Editor**

# CONTENTS

PREFACE....................................................................................................i
W. Jefferson Stroud, LaRC

## Part 1

OVERVIEW OF THE NASA PROGRAM IN COMPUTATIONAL STRUCTURAL MECHANICS ....................................1 S1
Murray Hirschbein, NASA Headquarters

## SESSION I - CONCURRENT PROCESSING METHODS AND APPLICATIONS

CSM PARALLEL STRUCTURAL METHODS RESEARCH..........................................25 S2
Olaf O. Storaasli, LaRC

COMPUTATIONAL STRUCTURAL METHODS AT NASA LEWIS.................................61 S3
L. J. Kiraly, LeRC

TRANSPUTER FINITE ELEMENT SOLVER...............................................................75 S4
Albert Danial, SPARTA

TRANSPUTER PARALLEL PROCESSING AT NASA LERC .......................................107 S5
Graham K. Ellis, ICOMP, LeRC

INNOVATIVE ARCHITECTURES FOR DENSE MULTI-MICROPROCESSOR COMPUTERS ..........................137 S6
Robert E. Larson, Expert-EASE

PARALLEL LINEAR EQUATION SOLVERS FOR FINITE ELEMENT COMPUTATIONS...............................................171 S7
James M. Ortega, Gene Poole, Courtenay Vaughan, Andrew Cleary, Brett Averick, U. of Virginia

ALGORITHMS AND SOFTWARE FOR SOLVING FINITE ELEMENT EQUATIONS ON SERIAL AND PARALLEL ARCHITECTURES..........................................203 S8
Alan George, U. of Tennessee

PARALLEL EIGENVALUE EXTRACTION.....................................................................239 S9
Fred A. Akl, Ohio U.

PARALLEL ALGORITHMS AND ARCHITECTURES FOR COMPUTATIONAL STRUCTURAL MECHANICS..........................................261 S10
Merrell Patrick, Shing Ma, Umesh Mahajan, Duke U.

THE FORCE: A PORTABLE PARALLEL PROGRAMMING LANGUAGE SUPPORTING COMPUTATIONAL STRUCTURAL MECHANICS ................................273 S11
Harry F. Jordan, Muhammad S. Benten, Juergen Brehm, Aruna Ramanan, U. of Colorado

METHODS FOR DESIGN AND EVALUATION OF PARALLEL COMPUTING SYSTEMS (THE PISCES PROJECT) .......... 512 281
Terrence W. Pratt, Robert Wise, Mary Jo Haught, U. of Virginia and ICASE, LaRC

MULTIPROCESSOR ARCHITECTURE: SYNTHESIS AND EVALUATION .......... 513 299
Hilda M. Standley, U. of Toledo

ENVIRONMENTAL CONCEPT FOR ENGINEERING SOFTWARE ON MIMD COMPUTERS .......... 514 323
L. A. Lopez and K. Valimohamed, U. of Illinois

HIERARCHIAL PARALLEL COMPUTER ARCHITECTURE DEFINED BY COMPUTATIONAL MULTIDISCIPLINARY MECHANICS .......... 355
Joe Padovan, Doug Gute, Keith Johnson, U. of Akron

## Part 2*

## SESSION II - ADVANCED METHODS & TESTBED/SIMULATOR DEVELOPMENT

CSM RESEARCH: TESTBED DEVELOPMENT .......... 387
Ronnie E. Gillian, LaRC

CSM TESTBED ARCHITECTURE .......... 419
Philip Underwood, Lockheed PARL

COMPUTATIONAL STRUCTURAL MECHANICS ENGINE STRUCTURES COMPUTATIONAL SIMULATOR .......... 459
C. C. Chamis, LeRC

INTERFACING MODULES FOR INTEGRATING DISCIPLINE SPECIFIC STRUCTURAL MECHANICS CODES .......... 487
Ned M. Endres, LeRC

CSM RESEARCH: METHODS AND APPLICATION STUDIES .......... 521
Norman F. Knight, Jr., LaRC

GENERIC ELEMENT PROCESSOR (APPLICATION TO NONLINEAR ANALYSIS) .......... 571
Gary Stanley, Lockheed PARL

ASSESSMENT OF SPAR ELEMENTS AND FORMULATION OF SOME BASIC 2-D AND 3-D ELEMENTS FOR USE WITH TESTBED GENERIC ELEMENT PROCESSOR .......... 653
Mohammad A. Aminpour, AS&M, LaRC

DEVELOPMENT AND VERIFICATION OF LOCAL/GLOBAL ANALYSIS TECHNIQUES FOR LAMINATED COMPOSITES .......... 683
O. Hayden Griffin, Jr., VPI&SU

CONTROL OF THE ERRORS OF DISCRETIZATION AND IDEALIZATION IN FINITE ELEMENT ANALYSIS .......... 733
Barna A. Szabó, Washington U.

* Published under separate cover

## Part 3*

BOUNDARY ELEMENTS FOR STRUCTURAL ANALYSIS .......... 763
Ray Wilson, P&W

DEVELOPMENT OF AN INTEGRATED BEM FOR HOT FLUID-STRUCTURE INTERACTION .......... 831
P. K. Banerjee and G. F. Dargush, SUNY-Buffalo

PROBABILISTIC STRUCTURAL ANALYSIS METHODS FOR SELECT SPACE PROPULSION SYSTEM STRUCTURAL COMPONENTS .......... 865
T. A. Cruse, SWRI

PROBABILISTIC FINITE ELEMENTS (PFEM) APPLIED TO STRUCTURAL DYNAMICS AND FRACTURE MECHANICS .......... 903
Wing-Kam Liu, Ted Belytschko, A. Mani, G. Besterfield, Northwestern U.

3-D INELASTIC ANALYSES FOR COMPUTATIONAL STRUCTURAL MECHANICS .......... 943
D. A. Hopkins and C. C. Chamis, LeRC

SPECIALTY FUNCTIONS FOR SINGULARITY MECHANICS PROBLEMS .......... 981
Nesrin Sarigul, Ohio State U.

### SESSION III - COMPUTATIONAL DYNAMICS

LARC COMPUTATIONAL STRUCTURAL DYNAMICS OVERVIEW .......... 1013
J. M. Housner, LaRC

ALGORITHMS AND SOFTWARE FOR NONLINEAR STRUCTURAL DYNAMICS .......... 1043
Ted Belytschko, Noreen D. Gilbertsen, Mark O. Neal, Northwestern U.

CONCURRENT ALGORITHMS FOR TRANSIENT FE ANALYSIS .......... 1067
M. Ortiz, Brown U. and B. Nour-Omid, Lockheed PARL

COMPUTATIONAL METHODS AND SOFTWARE SYSTEMS FOR DYNAMICS AND CONTROL OF LARGE SPACE STRUCTURES .......... 1105
K. C. Park, C. A. Felippa, Charbel Farhat, J. D. Downer, J. C. Chiou, W. K. Belvin, U. of Colorado

MULTI-GRID FOR STRUCTURES ANALYSIS .......... 1133
Albert F. Kascak, LeRC

---

* Published under separate cover

N89
29774

UNCLAS

# OVERVIEW OF THE NASA PROGRAM IN COMPUTATIONAL STRUCTURAL MECHANICS

Murray Hirschbein

NASA Headquarters

N89-29774

NC452981

S1-39

236.

311942

P. 23

# NASA COMPUTATIONAL STRUCTURAL MECHANICS

- DEVELOP ADVANCED ANALYTICAL/COMPUTATIONAL METHODS
- EXPLOIT NEWEST AND MOST POWERFUL COMPUTER TECHNOLOGY
- DIRECTED TOWARD VERY LARGE, COMPUTATIONALLY INTENSIVE, MEMORY EXTENSIVE PROBLEMS
- COMPUTATIONAL TESTBEDS TO SERVE AS TECHNOLOGY "INTEGRATORS" TO PROMOTE/ACCELERATE METHODOLOGY RESEARCH AND DEVELOPMENT

# NASA Computational Structural Mechanics

NASA has been active in developing computational structural analysis methodology for many years. However, these efforts were typically conducted as elements of broader research and technology programs without a central disciplinary focus. The purpose of developing a formal Computational Structural Mechanics (CSM) program was to provide this focus. This is likely to become particularly important in view of possible revolutionary advances in computing power that may occur with the development of ultra-fast concurrent processor computers with very large internal memory capacities (e.g., 256 Mflops/processor and 256 Mwords of memory for the Cray-2).

In order to meet the anticipated needs in modeling and analysis of advanced aerospace structures, NASA has developed a program focused on computational structural mechanics. The objective of this program is to advance the state-of-the-art in computational analysis to make accurate analysis of very large and complex structural problems routine. This will be accomplished by emphasizing two key areas: (1) the development of advanced analytical methods, extending beyond traditional approaches and, (2) the exploitation of the newest and most powerful parallel/multiprocessor computers available. Computational testbeds will be developed to serve as technology integrators and to promote/accelerate methodology research and development. An additional, and highly desirable, effect of the CSM program would be to influence the design of future hardware and software systems to reflect the needs of structural analysis.

# "CSM HISTORY"

## LANGLEY

- EXPLORATORY WORKSHOP, APRIL 1984
- PROGRAM ESTABLISHED OCTOBER 1984
  - 12 PROFESSIONALS IN CSM TEAM
  - 7 GRANTS
  - HEAVY EMPHASIS PLACED ON ADVANCED COMPUTING METHODS
  - COMMITMENT TO TESTBED
- TWO PROGRAM DEVELOPMENT WORKSHOPS
  - SEPTEMBER 1984 (INDUSTRY)
  - JUNE 1985 (UNIVERSITY)
- CURRENT STATUS
  - 18 PROFESSIONALS IN CSM TEAM
  - 15 GRANTS
  - 1 MAJOR SUPPORT CONTRACT
  - INITIAL VERSION OF TESTBED OPERATIONAL

## LEWIS

- EVOLVED FROM ENGINE STRUCTURES PROGRAM STARTED 1979
- FORMALLY STARTED OCTOBER 1985
  - CONSOLIDATED ADVANCED ANALYSIS CAPABILITY DEVELOPED UNDER BASE R&T AND SYSTEMS TECHNOLOGY PROGRAMS
  - INITIATED PROGRAM IN ADVANCED COMPUTING METHODS
  - LONG RANGE GOAL: FULL ENGINE STRUCTURAL ANALYSIS FOR A MISSION
  - 4 PROFESSIONALS IN STRUCTURES DIV.
  - 2 GRANTS
  - 4 MAJOR ANALYSIS DEVELOPMENT CONTRACTS (SYSTEM TECHNOLOGY)
- CURRENT STATUS
  - 9 PROFESSIONALS IN STRUCTURES DIV.
  - 7 GRANTS
  - SEVERAL ANALYSIS CODES DELIVERED

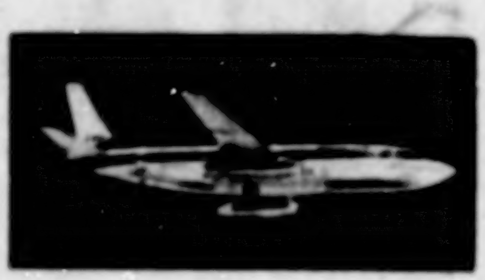
BOTH CENTERS NOW HAVE ACCESS TO A WIDE RANGE OF VERY ADVANCED COMPUTERS

# CSM History

Currently, the CSM effort is being actively pursued by two NASA centers. The Langley Research Center is focusing on airframe structures and large space structures while the Lewis Research Center is focusing on aeronautical and space propulsion structures. Both centers are building on a long history of activity in computational structural analysis and exploiting advanced computers. Lewis was one of the first NASA centers to obtain a supercomputer (first a Cray-1 then a Cray-XMP) and Langley was one of the first to have multi-processor computers (several years ago a specially designed "finite element machine" and more recently a Flex/32). The program is also being heavily supported by the use of the Cray-2 supercomputer (NAS) at the Ames Research Center and will continue to rely on the NAS facilities to provide the leading edge in computer technology.

# COMPUTATIONAL STRUCTURAL MECHANICS

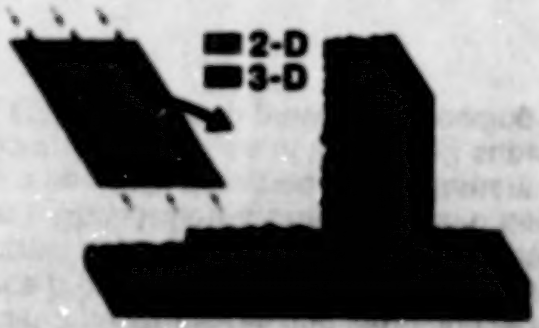
**Aircraft Structures**

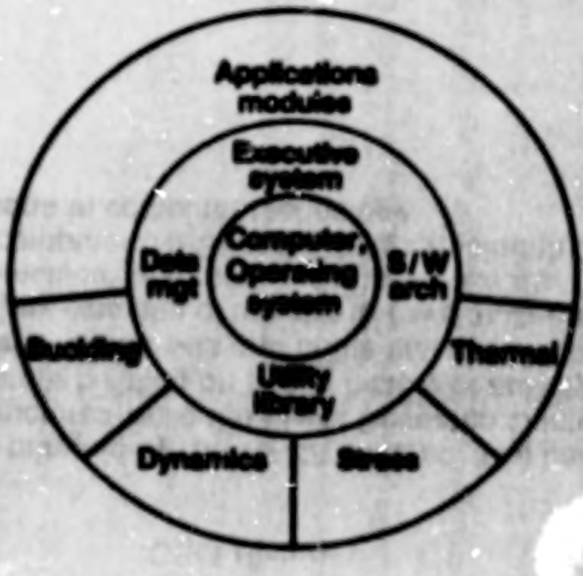


**Advanced Architecture Computers**

**Space Structures**

**Local 3-D nonlinear stress analysis within larger 2-D analysis model**





Applications modules

Executive system

Computer/ Operating system

Data mgt

S/W arch

Utility library

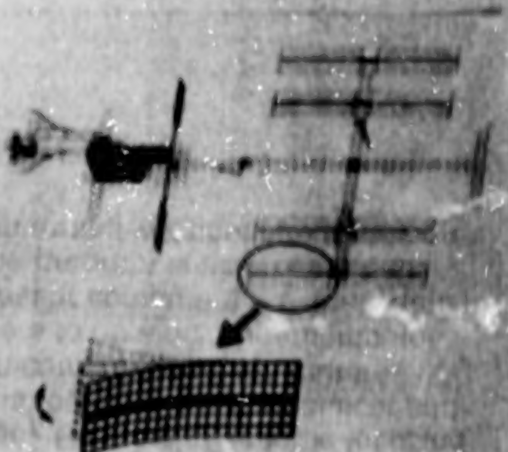
Buckling

Thermal

Dynamics

Stress

**Transient Dynamics**



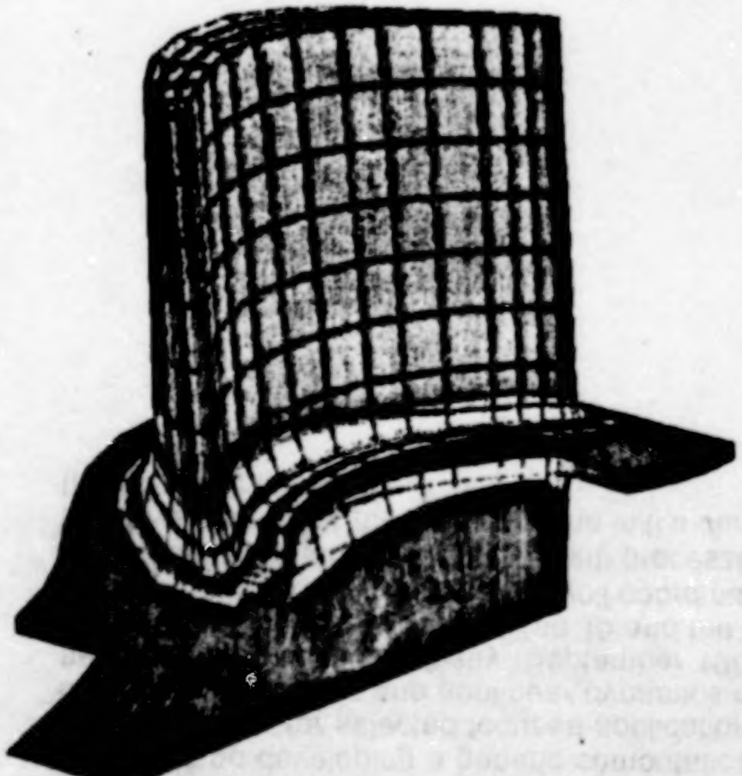
ORIGINAL PAGE
BLACK AND WHITE PHOTOGRAPH

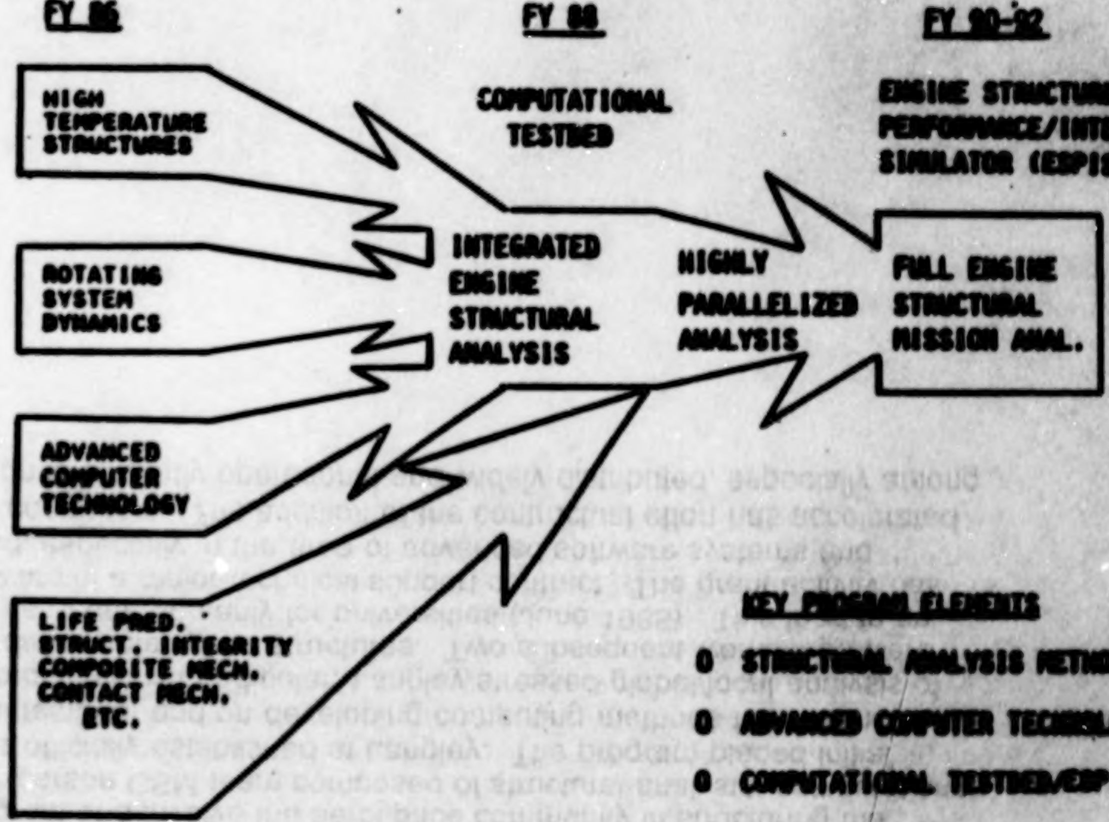
## Computational Structural Mechanics (LaRC)

The NASA CSM program was formally initiated at the Langley Research Center in 1984. An exploratory workshop was held to establish outside support for the program and involve the aerospace community in structuring the program. In October of 1984, an experienced 12-person CSM team composed of structural analysts, computational methods developers and computer scientists was officially established at Langley. The program placed initial emphasis on developing a generic computational testbed, and on developing computing methods for advanced computers and for selected focused application problems. In particular, Langley stressed global/local analysis of composite structures and nonlinear dynamics of deployable space structures. Two subsequent workshops were held, one primarily for industry (September 1984) and one primarily for universities (June 1985). This lead to an increase of funded grants from 7 to 15 and the award of a major technical support contract. The grant activity has proven to be very productive and well coordinated, especially in the area of advanced software systems and computational methods for concurrent processor computers. The addition of the contractual effort has accelerated the development of the testbed. The initial version is currently operational and widely distributed, especially among the CSM participants.

STRUCTURES DIVISION
ADVANCED CONCEPTS & APPLICATIONS BRANCH
NASA Lewis Research Center

# COMPUTATIONAL STRUCTURAL MECHANICS





FY 86

- HIGH TEMPERATURE STRUCTURES
- ROTATING SYSTEM DYNAMICS
- ADVANCED COMPUTER TECHNOLOGY
- LIFE PRED.<br>STRUCT. INTEGRITY<br>COMPOSITE MECH.<br>CONTACT MECH.<br>ETC.

FY 88

COMPUTATIONAL TESTBED

INTEGRATED ENGINE STRUCTURAL ANALYSIS

HIGHLY PARALLELIZED ANALYSIS

FY 90-92

ENGINE STRUCTURES PERFORMANCE/INTEGRITY SIMULATOR (ESPIS)

FULL ENGINE STRUCTURAL MISSION ANAL.

**KEY PROGRAM ELEMENTS**

- O STRUCTURAL ANALYSIS METHODS
- O ADVANCED COMPUTER TECHNOLOGY
- O COMPUTATIONAL TESTBED/ESPIS

ORIGINAL PAGE
BLACK AND WHITE PHOTOGRAPH

## Computational Structural Mechanics (LeRC)

While the CSM program at the Lewis Research Center "officially" began in 1985, Lewis had a well developed internal program in computational structural mechanics which was part of a broader engine structures program that began about 1978. Elements of these activities were consolidated into the CSM program which established a long-range goal of developing computational capability to enable detailed structural analyses of complete engine models to be conducted over complete mission cycles, routinely. The program emphasized development of advanced analytical methods, a strength of the engine structures program, and initiated an activity directed at exploiting advanced parallel/multi-processor computers. Initially, the CSM program was supported by the equivalent of 4 researchers. It funded 2 grants and benefited from the results of several on-going activities, including 4 major contracts to develop advanced analytical methods. Currently, there are effectively 9 researchers working on CSM at Lewis; 7 grants are funded; and the research contracts have delivered advanced computer codes covering inelastic finite element analysis, nonlinear boundary element analysis and probabilistic structural analysis.

# CURRENT MAJOR AREAS OF EMPHASIS

**LANGLEY**

- TESTBED DEVELOPMENT
- GLOBAL/LOCAL ANALYSIS
- NONLINEAR TRANSIENT DYNAMICS
- ADVANCED FINITE ELEMENT ANALYSIS
- PARALLEL PROCESSING
- APPLICATION STUDIES (AS A PART OF IN-HOUSE ACTIVITIES)

**LEWIS**

- ENGINE STRUCTURES COMPUTATIONAL SIMULATOR DEVELOPMENT
- INELASTIC FINITE ELEMENT ANALYSIS
- NONLINEAR BOUNDARY ELEMENT ANALYSIS
- PROBABILISTIC STRUCTURAL ANALYSIS
- NONLINEAR TRANSIENT DYNAMICS
- TRANSPUTER SYSTEMS

## Current Major Areas of Emphasis

The major areas of emphasis in the CSM program cover a broad spectrum of advanced analytical methods, computational methods and focused applications. At the center are computational testbeds intended for use by NASA, active CSM participants and by outside users. The testbeds will serve to integrate the methodology being developed across the program and provide access to NASA-developed technology by interested researchers. The generic testbed being developed by Langley is UNIX-based and is currently available on VAX, Cray and Flex computers within NASA and will be extended to other computers in the future. The initial version has been widely distributed among participants in the NASA CSM program and can be supplied, upon request, to other organizations wishing to use it. The testbed being developed at Lewis is more focused on applications related to engine structures and is not currently intended for distribution. However, it will be accessible to selected outside users through NASA.

Advanced analytical capability is being developed for transient and nonlinear dynamics, advanced finite element analysis, nonlinear boundary element analysis and probabilistic structural analysis and global/local analysis. All of these capabilities are currently being incorporated directly into the testbeds and/or exist as complete stand-alone computer codes. In addition, fundamental methods are being developed for conducting structural analyses on multi-processor computers, including eigenanalysis, solutions to systems of linear equations and basic matrix operations. These techniques will be incorporated into the structural analysis methods as they, in turn, are developed for concurrent processor computers. These methods also form part of the effort to make the testbeds easily adaptable to general classes of multi-processor computers.

# AREAS OF FOCUSED APPLICATION

- COMPOSITE STRUCTURES
  - MECHANICS AND FAILURE MECHANISMS
  - BUCKLED/POST-BUCKLED BEHAVIOR
- NONLINEAR HIGH-TEMPERATURE STRUCTURAL ANALYSIS
  - AIRCRAFT ENGINE STRUCTURES
  - SPACE SHUTTLE MAIN ENGINE
  - (HYPERSONICS AIRCRAFT STRUCTURES)
- SYSTEM DYNAMICS
  - LARGE AMPLITUDE TRANSIENT MOTION (DEPLOYABLE SPACE STRUCTURES)
  - HIGH SPEED (ROTATING) ENGINE STRUCTURES
- TIRE MECHANICS

## Areas of Focused Application

A strong motivating factor for developing the CSM program was the need for large-scale advanced analytical capability in several areas of activity within NASA including basic research and systems technology. These areas serve to focus the CSM activities on current problems which can benefit immediately from newly developed capability, provide realistic problems and experimental data with which to develop and validate new capabilities, and expose strengths and weaknesses to guide the program in the future.

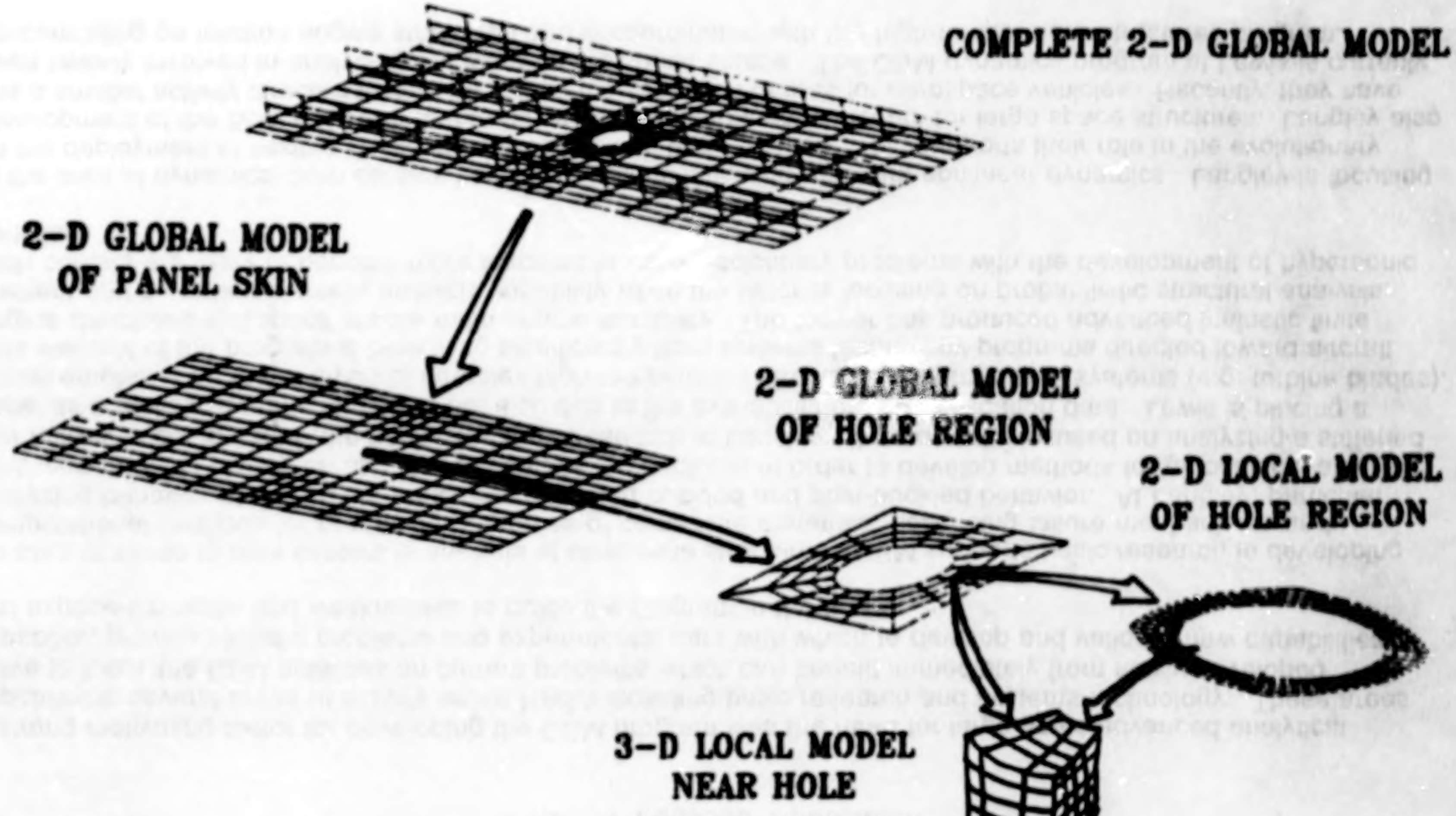
An area common to both centers is analysis of composite structures. CSM supports basic research in developing computational methods for predicting properties of composite materials, describing failure mechanisms and analyzing behavior of composite structures including buckled and post-buckled behavior. At Langley, particular emphasis is being placed on analysis of composite structures in order to develop methods for global/local analysis. The activity is coordinated with basic structures research at Langley. It is currently focused on analyzing a stiffened panel as a realistic generic structure, but also due to the availability of good validation data. Lewis is placing a strong emphasis on the analysis of complex high-temperature structures for propulsion systems (e.g. turbine blades). This element of the program is benefiting significantly from systems technology programs directed toward aircraft engine structures and space shuttle main engine durability. The former has produced advanced inelastic finite element and boundary element analysis capability while the latter is focusing on probabilistic structural analysis. Both centers are likely to become more involved in multi-disciplinary problems with the development of hypersonic vehicles.

In the area of dynamics, both centers have applications involving transient nonlinear dynamics. Langley is focusing on the deployment of flexible space structures such as large trusses. This supports their role in the evolutionary development of the Space Station and in developing fundamental concepts for large space structures. Langley also has a smaller activity directed toward analyzing the dynamics of tires for aerospace vehicles. Recently, they have been heavily involved in analyzing tire wear on the space shuttle. The CSM dynamics program at Lewis is currenly concentrating on rotation engine structures and is coordinated with the high-temperature structures program.

The applications of CSM technology and the areas of fundamental research are strongly influenced by the existing and long-range needs of NASA and the general aerospace community. The general approach has been, and will continue to be, to develop capability based on long-range needs but to emphasize applications to current relevant problem areas.

# GLOBAL/LOCAL ANALYSIS OF CSM FOCUS PROBLEM

## METHODS AND APPLICATIONS



COMPLETE 2–D GLOBAL MODEL

2–D GLOBAL MODEL
OF PANEL SKIN

2–D GLOBAL MODEL
OF HOLE REGION

2–D LOCAL MODEL
OF HOLE REGION

3–D LOCAL MODEL
NEAR HOLE

## Global/Local Analysis of CSM Focus Problem

An example of a focused CSM application problem at Langley is the stiffened composite panel with a hole. This structure is typical of aircraft structures and is being studied as part of the Langley structures program in order to understand and predict buckling/post-buckling behavior and failure mechanisms. The particular region of study is the area around the hole, especially near the discontinuity at the stiffener. The object of the CSM effort is to avoid large, highly detailed models of the entire structures, by developing methods to "converge" on the hole by moving from a relatively sparse global 2-D model, to a local 2-D model, to a "small" detailed 3-D model that accurately represents thickness effects. In following this process, it is extremely important to assure that each higher level model contains enough information for accurate transition to a more local model. This may require additional intermediate models and some iteration, but the goal of CSM is to assure a desired level of accuracy with a minimal level of computational effort and to do this in an automated manner requiring a minimal amount of intervention. This capability is being incorporated into the testbed and will also serve as a focused problem for developing concurrent processing methods. More significantly, it will be further integrated into a generic global approach to analyzing very large structural problems such as an entire airframe.

LEVELS OF SRB ANALYSIS

2-D shell model of SRB with equivalent joints
3-D solid mo
2-D shell model of joint

ORIGINAL PAGE
BLACK AND WHITE PHOTOGRAPH

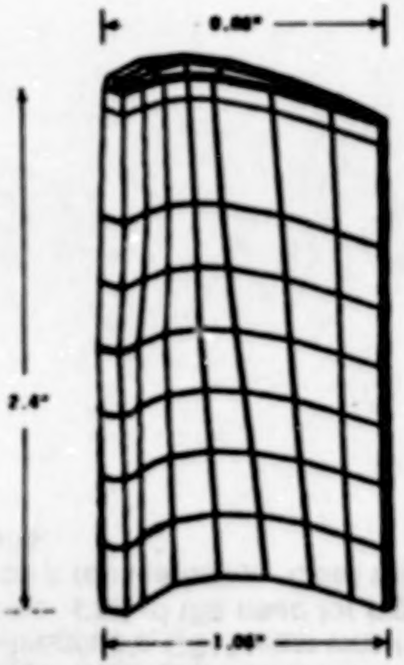
## Levels of SRB Analysis

Recently, the CSM group at Langley provided extensive support in advanced structural modeling and analysis to the NASA shuttle re-design effort presenting some of the most detailed analysis yet conducted on the shuttle solid rocket booster (SRB) cases. They produced and analyzed a highly detailed, 83000 degree-of-freedom shell model of the entire rocket motor; a more refined model of a section near the clevis-tang joint; and a highly refined 14000 degree-of-freedom, 3-D model of a narrow segment about the clevis joint including the shear pin. This last model can also account for some of the frictional nonlinearities that can arise from pin case contact and can be further used to transfer these effects to the full model of the SRB. This analysis required the full resources of a Cray-XMP.

This was a "crash" effort and used the best analytical tools currently available. Its significance is not so much in what was accomplished but in the amount of human and computational effort that was required to do it. The fact that a very knowledgeable CSM team was in place at Langley greatly contributed to the success of this task. However, it clearly demonstrated the need for highly streamlined, computationally efficient methods for attacking problems such as these on a routine basis. CSM will continue to develop, in part, by selecting problems such as these for focused applications.

# NONLINEAR STRUCTURAL ANALYSIS OF AN AIR-COOLED TURBINE BLADE

## (SIMPLIFIED)



SIMPLE STRUCTURAL MODEL
VERY COMPLEX MATERIAL MODEL
THERMAL AND CENTRIFUGAL LOADING

⇒

418 NODES
173 ELEMENTS
20 SEC./LOAD STEP
200 STEP/MISSION CYCLE
5 - 10 CYCLE/SIMULATION

⇓

5 - 10 HOURS ON CRAY-XMP

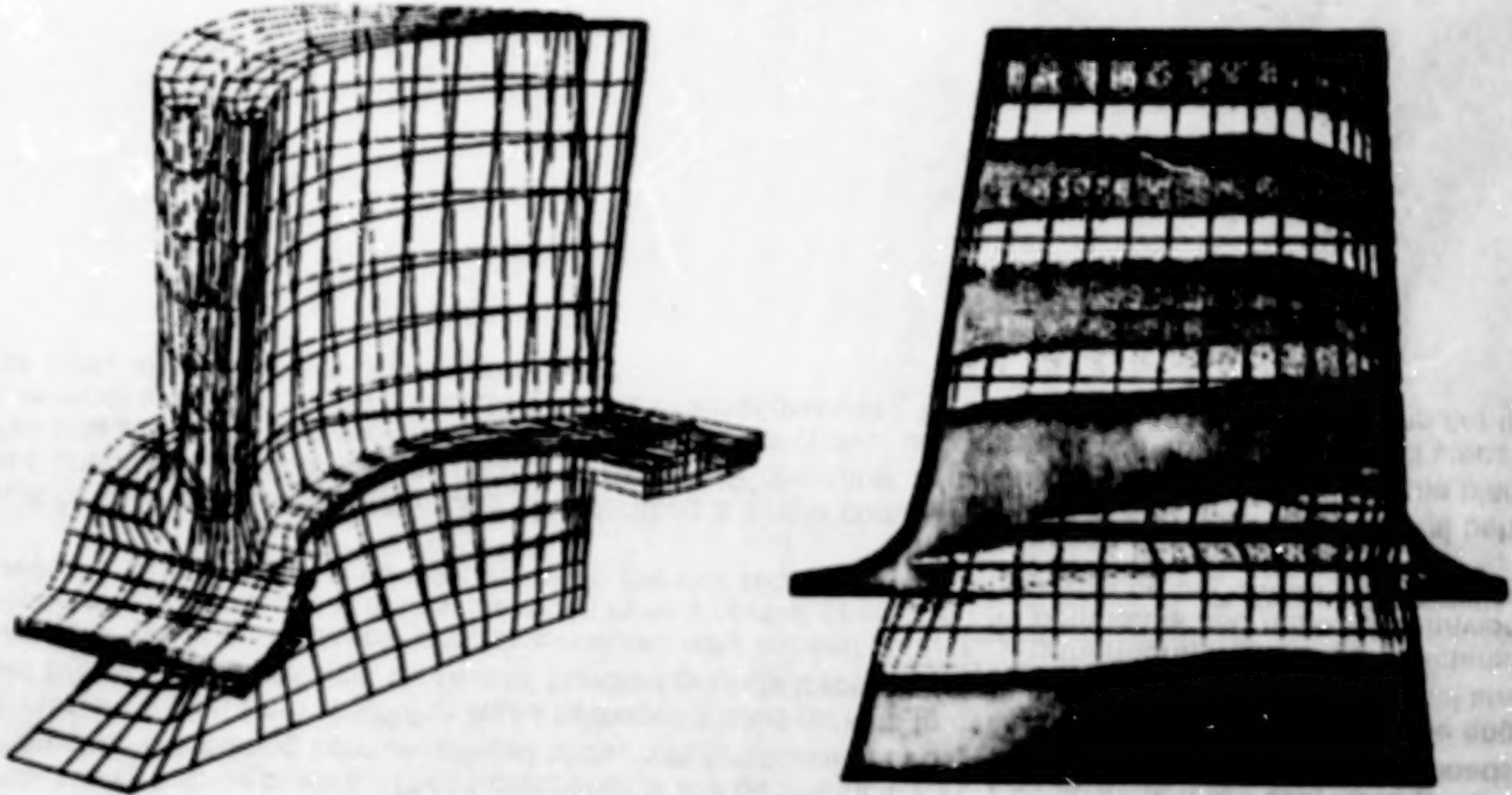
MORE REALISTIC MODELS - 2000 NODES, 1000 ELEMENTS
" DAYS OF COMPUTER TIME ? "

# Nonlinear Structural Analysis of an Air-Cooled Turbine Blade

An example of computational structural mechanics at Lewis Research Center is the analysis of advanced turbomachinery turbine blades. These components are geometrically very complex and are subjected to severe thermomechanical loading from centrifugal force, with rotational speeds as high as 36000 RPM; thermal loads, with temperatures ranging from -450°F in liquid hydrogen fueled rockets to over 2500°F in aircraft gas turbine engines; and gas pressure loads as high as several hundred pounds force. In future propulsion systems, rotational speeds and pressure forces could double and temperatures may exceed 4000°F. Furthermore, materials characteristics can become very nonlinear in critical locations during normal operation and both loads and material behavior can vary over the component in a non-deterministic manner requiring that the analysis be performed probabilistically.

Analysis of a simplified structural model incorporating a single complex load case and nonlinear material behavior requires about 20 sec. of computing time on a Cray-XMP computer. However, a complete analysis of the blade requires that it be analyzed over several simulated mission cycles, consisting of 100-200 separate load steps per cycle, in order to determine the cumulative effects of extended use. This would take 5-10 hours of computer time and, as such, analysis of this type is rarely done.

SSME HPFTP - FIRST STAGE TURBINE BLADE
SINGLE CRYSTAL BLADE DYNAMICS
MODE 1 - FREQUENCY 4487 HZ



ORIGINAL PAGE
BLACK AND WHITE PHOTOGRAPH

## SSME HPFTP - First Stage Turbine Blade

The more realistic model of a turbine blade shown contains 1025 solid elements and 1575 nodes and is representative of turbine blades in rocket engine turbopumps and aircraft gas turbine engines. (Actually, modern aircraft gas turbine blades have internal cooling passages and can be much more complex than the model shown.) Depending on the severity of the loads, a complete structural analysis for a single loads case requires about 1-5 minutes of computer time on a Cray-XMP and is performed routinely. However, a detailed analysis of this blade model over several mission cycles could take well over 100 hours of computer time. The goal of CSM at Lewis is to make analysis of this type routine and, ultimately, to extend computational capability to the point of analyzing entire engine configurations in detail. Projections of increases in computing power over the next ten years indicate that this is likely, but it will have to be done by developing new, innovative approaches to basic structural analysis as well as exploiting the most powerful computers available. The computational testbed will facilitate this process and the analytical methods currently under development are the first step toward this goal.

# SIGNIFICANT NASA COMPUTERS AVAILABLE TO CSM

- CRAY-2 – NUMERICAL AERODYNAMIC SIMULATOR (NAS-AMES)
- CRAY-XMP – LEWIS AND AMES
- FLEX/32 – LANGLEY
- TRANSPUTER SYSTEM (67 PROCESSORS) – LERC (ALSO POSSIBLY LARC)
- RIAC (RESEARCH INSTITUTE FOR ADVANCED COMPUTING) – AMES
  - CONVEX C1
  - SEQUENT BALANCE 2000
  - ALLIANT (TO BE DELIVERED)
  - INTEL ISPC HYPERCUBE
  - CONNECTION MACHINE CM2 (TO BE DELIVERED)
  - DATA FLOW MACHINE (TO BE DELIVERED)
  - PRINCETON "NAVIER-STOKES" MACHINE (TO BE DELIVERED)
- PROPOSED SUPER-MINICOMPUTERS – LANGLEY AND LEWIS
- AMDAHL MAINFRAME COMPUTERS – LEWIS AND AMES
- IBM 3033 – LEWIS
- VPS-32 – LANGLEY
- SEVERAL VAX
- MPP – GODDARD

## Significant NASA Computers Available to CSM

NASA has a wide range of advanced computers available to the CSM program, including Cray-2 and Cray-XMP supercomputers and a variety of the most modern concurrent processing computers. These are in addition to powerful computer resources available at Langley and Lewis. Furthermore, the NAS computer facility at Ames will continue to provide the most powerful computers available by maintaining state-of-the-art supercomputers. In the next few years, the Cray-2 will become the "second" computer at the center when a newer one, at least 4 times faster, is acquired. This process is planned to be repeated every 2 years thereafter with at least 2 generations of supercomputers available concurrently. Also, the Research Institute for Advanced Computing (RIAC) at Ames will maintain an array of advanced concurrent processor computers. While RIAC is intended as a part of a broader computer science research activity its computers are available to CSM researchers through the "NASnet" networking system. These extensive resources are considered to be one of the major strengths of the CSM program. They will allow researchers to work on an exceptionally broad range of computer architectures and be assured ready access to the most powerful supercomputers.

N89
29775
UNCLAS

N91-29775

52-39
356
311943
P-35

# CSM PARALLEL STRUCTURAL METHODS RESEARCH

Olaf O. Storaasli *

NASA Langley Research Center

## INTRODUCTION

For many years, the Structures and Dynamics Division at NASA Langley has conducted research on the forefront of parallel and high-speed scientific computing by developing innovative software and hardware to speed up finite element structural analysis computations.

Ten years ago, using some of the first microcomputers, Langley researchers performed some of the first, if not the first, parallel structural analysis computations using four computers (connected together and coded in assembly language) to solve the finite element beam bending equations. This early experience subsequently led to the development by Langley of the Finite Element Machine (ref. 5), a more comprehensive parallel computer with up to 36 processors. With this parallel computer, significant reductions in computation time were achieved for widespread applications including matrix equation solution, dynamic transient analysis, eigensolution and nonlinear elasto-plastic yield surface computations.

When the first commercial parallel computers appeared, Langley purchased a 20-processor FLEX/32 and began Computational Structural Mechanics (CSM) parallel methods research on it.

This presentation summarizes our CSM Parallel Structural Methods Research and provides an introduction for six members of our research team who will speak today (Drs. Robert Larson, Jim Ortega, Alan George, Harry Jordan, Terry Pratt and Merrell Patrick) and Phil Underwood who will speak tomorrow.

---

* Aerospace Engineer, Structural Mechanics Branch, Structures and Dynamics Division.

PRECEDING PAGE BLANK NOT FILMED

PAGE 24 INTENTIONALLY BLANK

N89-29775

# LANGLEY CSM PROGRAM

**Parallel Structural Methods Research - Olaf O. Storaasli**

**Testbed Development - Ronnie Gillian**

**Methods and Applications Studies - Norm Knight**

# LANGLEY CSM PROGRAM

The Langley Computational Structural Mechanics (CSM) program consists of the three primary components shown on this slide. Today we will discuss the work of the Parallel Structural Methods Research team that I lead. Tomorrow the Testbed and Methods and Applications work led by Ronnie and Norm will be discussed.

In addition to these primary CSM research thrusts, some cooperative work is being conducted with the Structural Dynamics Branch which will be addressed Friday morning by Dr. Jerrold Housner.

# OUTLINE

**Objective and Approach**

**Team Research Strategy**

**Parallel Architectures and Software**

**CSM Focus Problems**

**Typical Results**

**Future Directions**

## OUTLINE

This talk is organized to address the six items shown on this outline. These items should cover the major aspects of our research work as well as provide a suitable background and introduction for the other team members who speak after me. Although I will present a sample of some of our typical results in selected areas, other team members will provide more detailed results from their specific research area.

# PARALLEL STRUCTURAL METHODS

**Objective:** To develop structural analysis methods for parallel computers.

**Approach:** Design, develop and implement computational utilities, solution methods and languages for a parallel processing environment.

Evaluate and compare parallel methods by solving CSM focus problems.

Incorporate methods in testbed software.

# PARALLEL STRUCTURAL METHODS

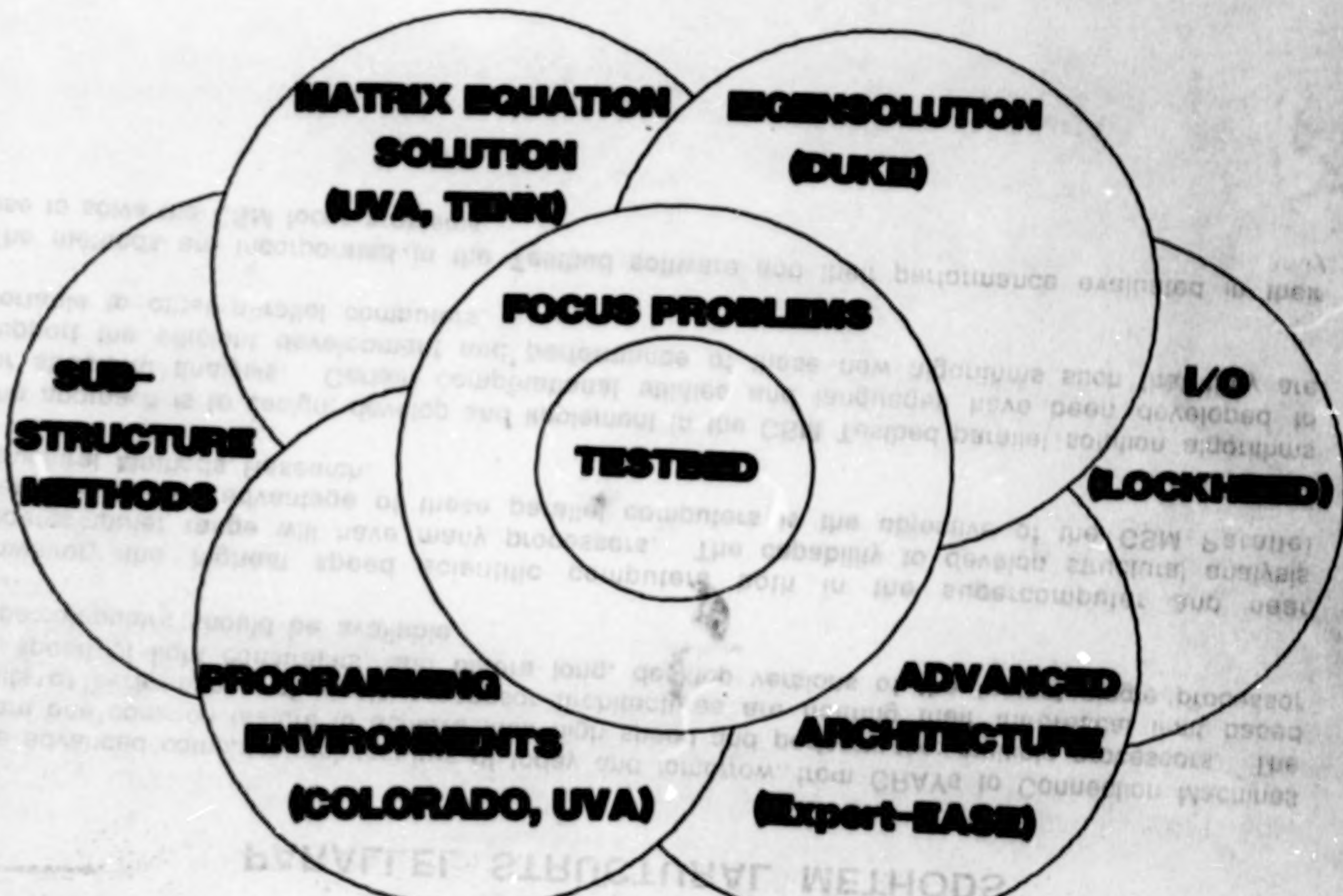
The advanced computer architectures of today and tomorrow, from CRAYs to Connection Machines share one common feature to achieve their high speed and performance: multiple processors. The limits of performance of single processor architectures are nearing their theoretical limit based on speed of light constraints, and before long, desktop versions of the fastest single processor supercomputers should be available.

However, the highest speed scientific computers both in the supercomputer and near supercomputer range will have many processors. The capability to develop structural analysis methods to take advantage of these parallel computers is the objective of the CSM Parallel Structural Methods Research.

The approach is to design, develop and implement in the CSM Testbed parallel solution algorithms for structural analysis. Certain computational utilities and languages have been developed to support the efficient development and performance of these new algorithms such that they are portable to other parallel computers.

The methods are incorporated in the Testbed software and their performance evaluated in their use to solve the CSM focus problems.

# PARALLEL METHODS RESEARCH TEAM



MATRIX EQUATION SOLUTION (UVA, TENN)

EIGENSOLUTION (DUKE)

SUB-STRUCTURE METHODS

I/O (LOCKHEED)

FOCUS PROBLEMS

TESTBED

PROGRAMMING ENVIRONMENTS (COLORADO, UVA)

ADVANCED ARCHITECTURE (Expert-EASE)

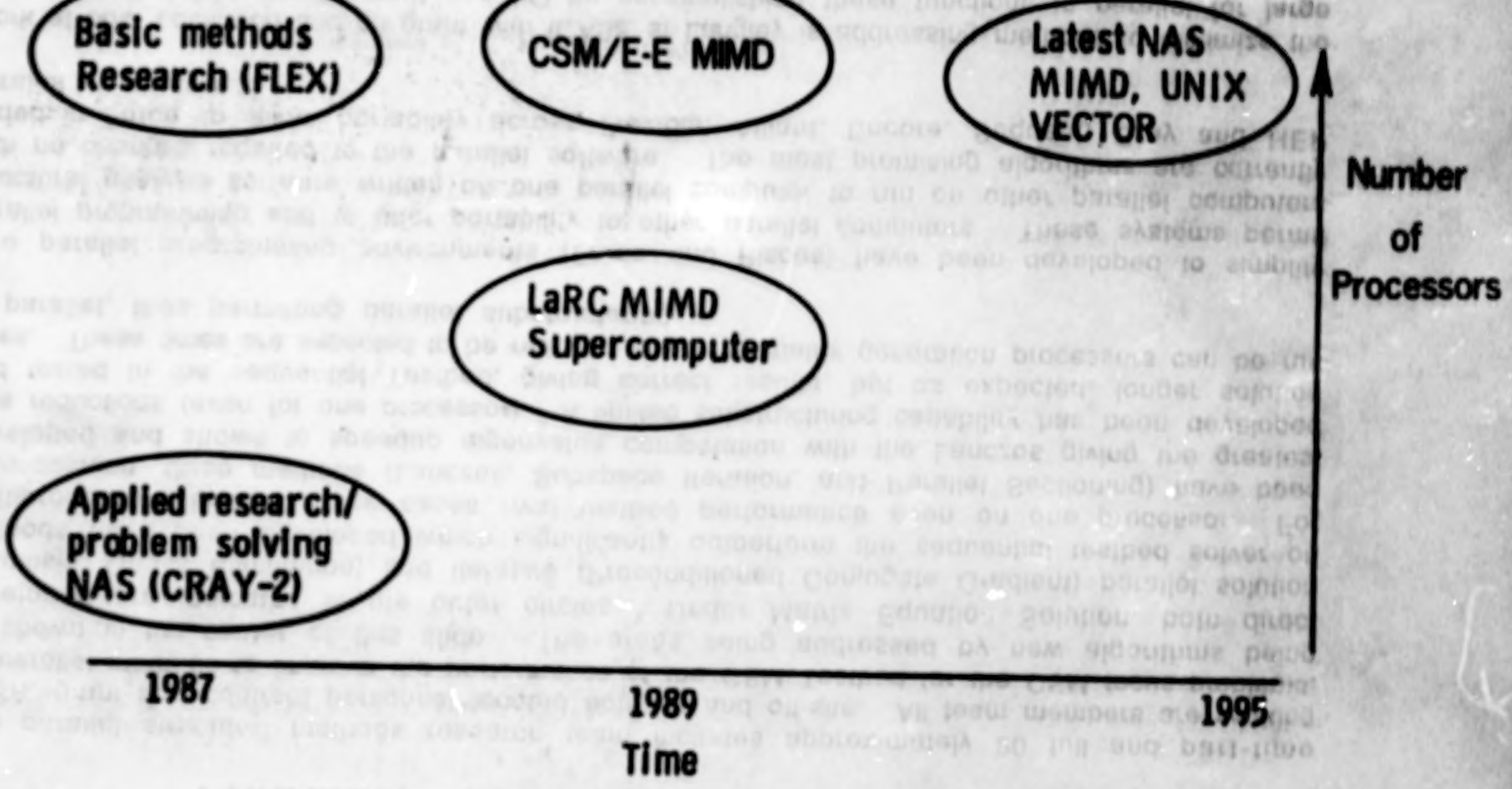
# PARALLEL METHODS RESEARCH TEAM

The parallel structural methods research team includes approximately 20 full and part-time NASA, grant and contract personnel located both on and off-site. All team members are working on parallel methods to improve the performance of the CSM Testbed for the CSM focus problems, as shown in the center of this slide. The areas being addressed by new algorithms being developed are indicated in the outer circles. Under Matrix Equation Solution, both direct (Choleski, Gauss Elimination) and iterative (Preconditioned Conjugate Gradient) parallel solution methods have been developed which significantly outperform the sequential testbed solver on multiprocessors and in some cases rival testbed performance even on one processor. For Eigensolution, three methods (Lanczos, Subspace Iteration, and Parallel Sectioning) have been developed and shown to speedup eigenvalue computation with the Lanczos giving the greatest time reductions (even for one processor). A limited substructuring capability has been developed and tested in the sequential Testbed, giving correct results, but as expected, longer solution times. These times are expected to be reduced when the matrix generation processors can be run in parallel, thus permitting parallel substructuring.

Two parallel programming environments (Force and Pisces) have been developed to simplify parallel programming and to offer portability to other parallel computers. These systems permit structural analysis software written on one parallel computer to run on other parallel computers with no changes required to the parallel software. The most promising algorithms are currently coded in Force to allow portability across Flexible, Alliant, Encore, Sequent, Cray and HEP parallel computers.

Work at both Lockheed and on grant with ICASE at Langley is addressing methods to minimize the time spent for data management and I/O by accomplishing these functions in parallel for large finite element applications. Finally, an advanced architecture parallel computer design based on a chordal ring of Inmos T-800 processors is planned for delivery to CSM in 1989. It should contain at least 15 processors each with a 64-bit floating point unit and a peak performance of approximately 90 million Whetstones for a total system peak performance of 34 MFLOPS.

# ADVANCED ARCHITECTURE COMPUTERS FOR CSM PARALLEL STRUCTURAL METHODS RESEARCH



Basic methods Research (FLEX)

CSM/E-E MIMD

Latest NAS MIMD, UNIX VECTOR

LaRC MIMD Supercomputer

Applied research/ problem solving NAS (CRAY-2)

Number of Processors
1987
1989
1995
Time

# ADVANCED ARCHITECTURE COMPUTERS FOR PARALLEL CSM RESEARCH

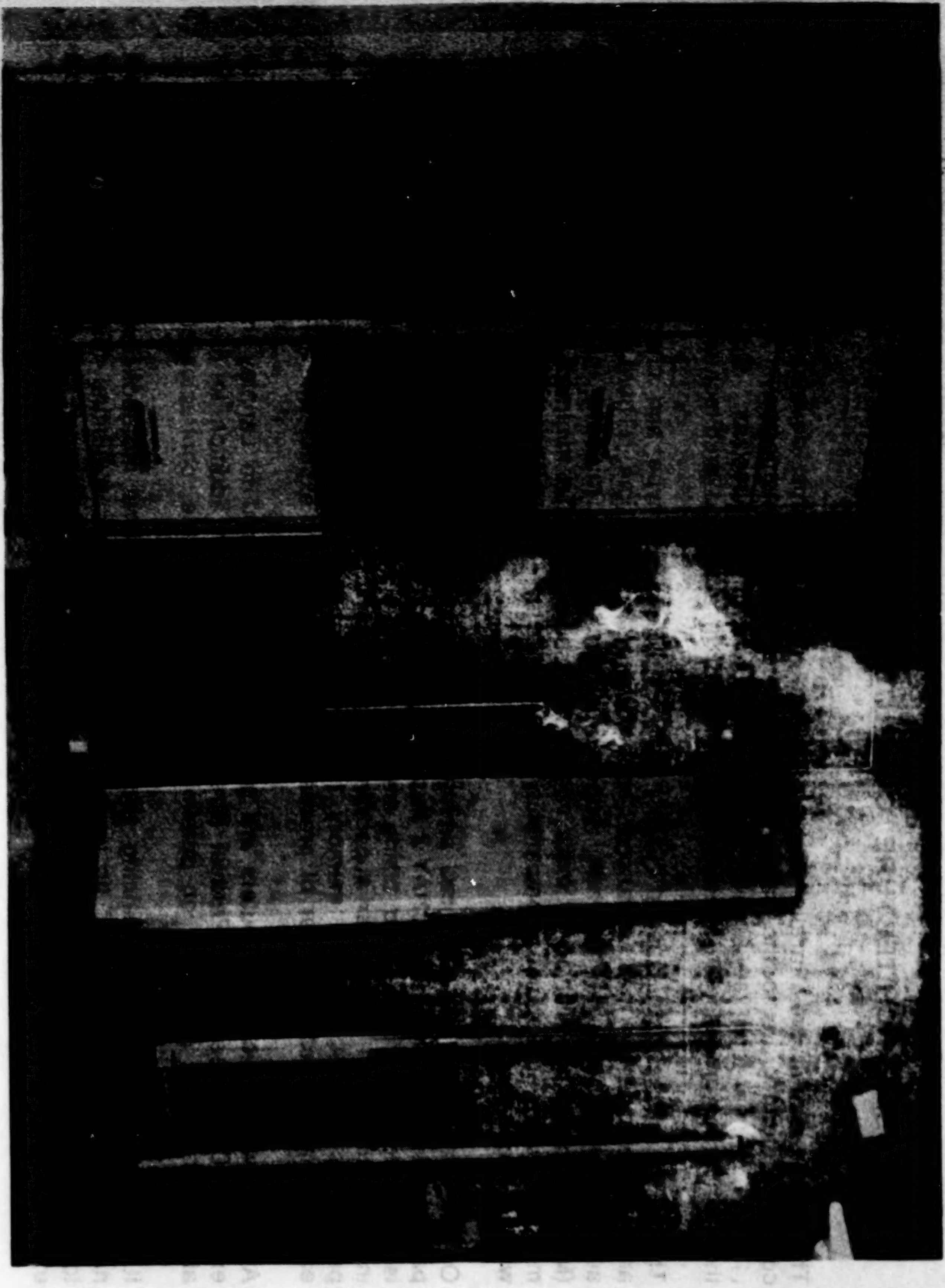
The intent is for the Parallel Structural Analysis Algorithms developed on today's parallel computers to migrate to the parallel and supercomputers of tomorrow. This slide shows today (1987), the near-term (1989) and the far-term future on the abscissa. On the ordinate is shown the number of processors (or level of parallelism) in advanced architecture computers.

The circle on the right labelled "Latest NAS" indicates the type and characteristics of what is anticipated to be the most advanced supercomputer at that time. There is little question that to achieve high-speed performance, supercomputers will be multiple instruction multiple data (MIMD) architecture with 16 or more vector processing units. To minimize development costs and maximize user acceptance, compatibility and portability, the UNIX operating system and utilities will be required.

On the left are the two advanced architecture multiprocessor computers primarily used in CSM parallel methods work. The CRAY-2 has four processors with a single path to shared memory and is useful for vectorization and timing studies and may become more useful for parallel research in the future when a compiler supporting parallel constructs is available. The FLEX/32 with 20 processors and 84.5 MB of memory (both local and global) supports research on algorithms exploiting a significant number of processors (what we might expect with CRAY in the 1990s).

At least two additional computers are planned to be delivered to Langley in 1989 to significantly enhance CSM research in Parallel Structural Methods. In addition to these, certain testing of algorithms is also performed on parallel computers at grantee, contractor and other sites.

It is expected that by maintaining the capability to explore methods exploiting a significant number of processors as well as implementing on computers exhibiting the maximum speed for today, we shall be in a position to have algorithms with the proper characteristics to run most efficiently on the fastest scientific computers in the future.



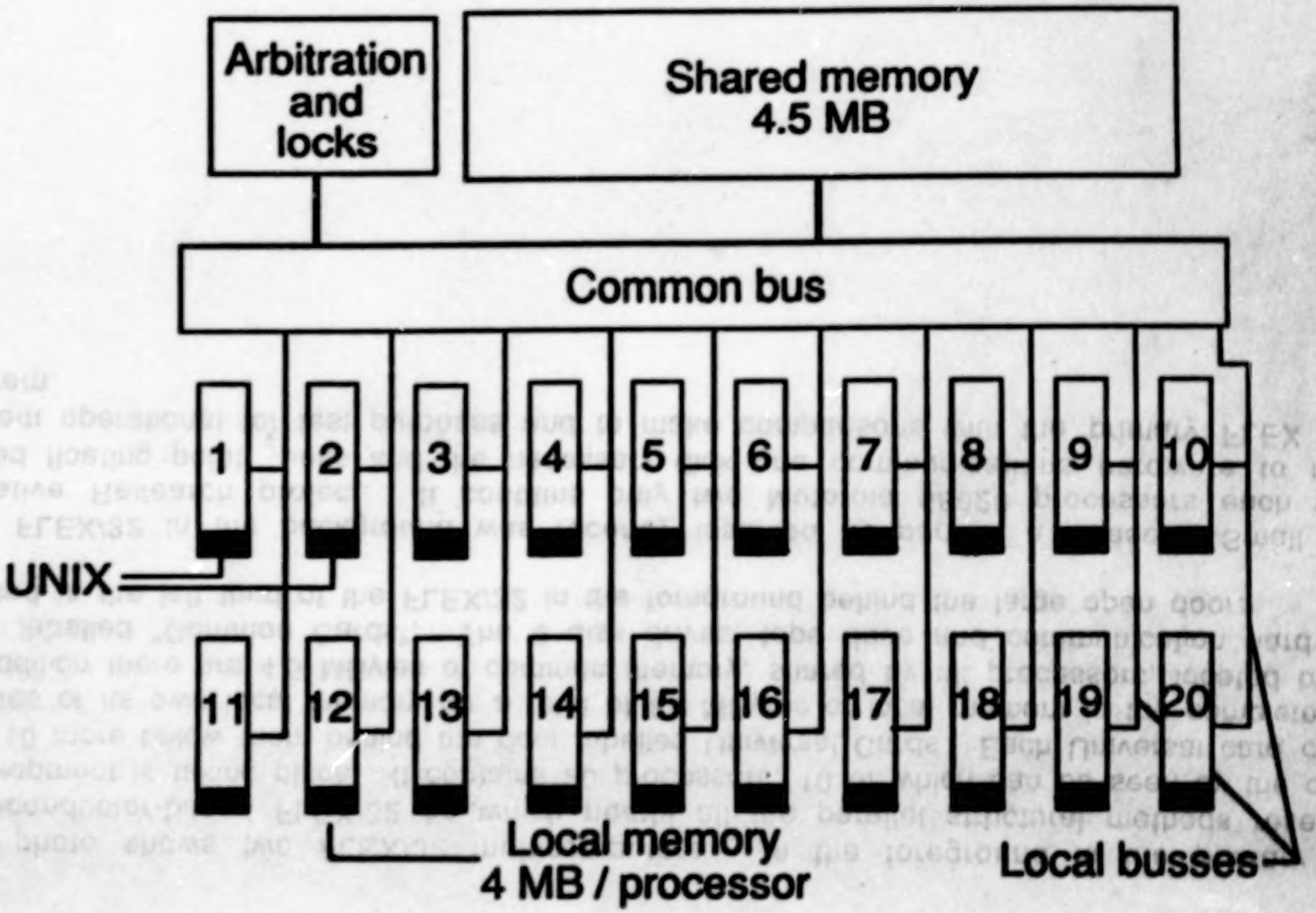
ORIGINAL PAGE
BLACK AND WHITE PHOTOGRAPH

# FLEX/32 MULTICOMPUTER

This photo shows two FLEX/32 multicomputers. In the foreground is the primary National Semiconductor-based FLEX/32 on which nearly all the parallel structural methods research and development is taking place. It contains 20 processors, 10 of which can be seen by the open door and 10 more below them behind the door labelled Universal Cards. Each Universal card contains 4 MBytes of its own local memory for a total of 80 MBytes of local memory in the complete system. In addition there are 4.5 MBytes of common memory, shared by all processors, located behind the door labelled "Common Cards". The 8 disk drives, tape drive and communication hardware are located in the left third of the FLEX/32 in the foreground behind the large open door.

The FLEX/32 in the background was recently installed as part of a Phase 2 Small Business Initiative Research project. It contains only two Motorola 68020 processors each with high speed floating point units and the necessary disk and communications hardware to make the system operational for test purposes and to make comparisons with the primary FLEX computer system.

# FLEX/32 20-PROCESSOR CONFIGURATION


Arbitration and locks
Shared memory 4.5 MB
Common bus
1
2
3
4
5
6
7
8
9
10
UNIX
11
12
13
14
15
16
17
18
19
20
Local memory 4 MB / processor
Local busses

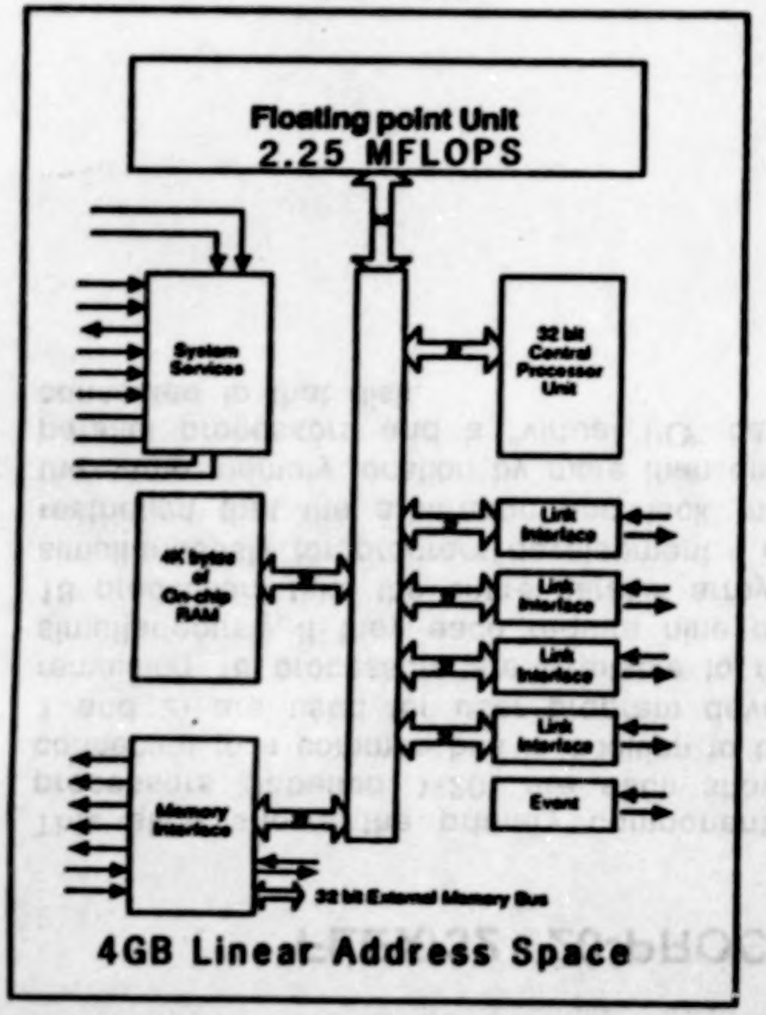
## FLEX/32 20-PROCESSOR CONFIGURATION

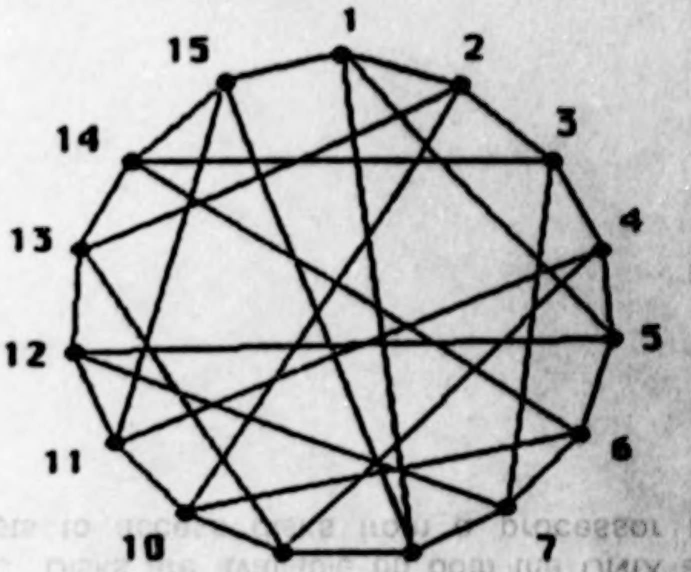
This slide shows the primary components of the primary FLEX/32 multicomputer. The 20 processors (labelled 1-20) are each shown to contain 4MB memory (black shading) and are connected to a common bus in addition to being paired via local busses. Two processors (labelled 1 and 2) are used for user program development using the UNIX operating system while the remaining 18 processors are available to run parallel programs. Two parallel programs may run simultaneously if they each require nine or fewer processors and programs requiring from 9 to 18 processors take the entire parallel array, although the two UNIX processors may still be used simultaneously for program development. All processors can access the shared memory with the restriction that the arbitration and lock mechanism used prevents simultaneous contention for the same memory location by more than one processor. Disks are available on both the UNIX and parallel processors and a "virtual I/O" capability exists to access disks from a processor not connected to that disk.

# MULTI-MICROPROCESSOR COMPUTER ARCHITECTURE

## - FAMILY OF LOW-COST SUPERCOMPUTERS -

IMS T800


Floating point Unit
2.25 MFLOPS
System Services
32 bit Control Processor Unit
4K bytes of On-chip RAM
Link Interface
Link Interface
Link Interface
Link Interface
Event
Memory Interface
32 bit External Memory Bus
4GB Linear Address Space


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

| processor | | Whetstones/second single length |
|---|---|---|
| Intel 80286/80287 | 8 MHz | 300K |
| IMS T414-20 | 20 MHz | 663K |
| NS 32332-32081 | 15 MHz | 728K |
| MC 68020/68881 | 16/12 MHz | 755K |
| ATT 32000/32100 | | 1000K |
| Fairchild Clipper | 33 MHz | 2220K |
| IMS T800-20 | 20 MHz | 4000K |
| IMS T800-30 | 30 MHz | 6000K |

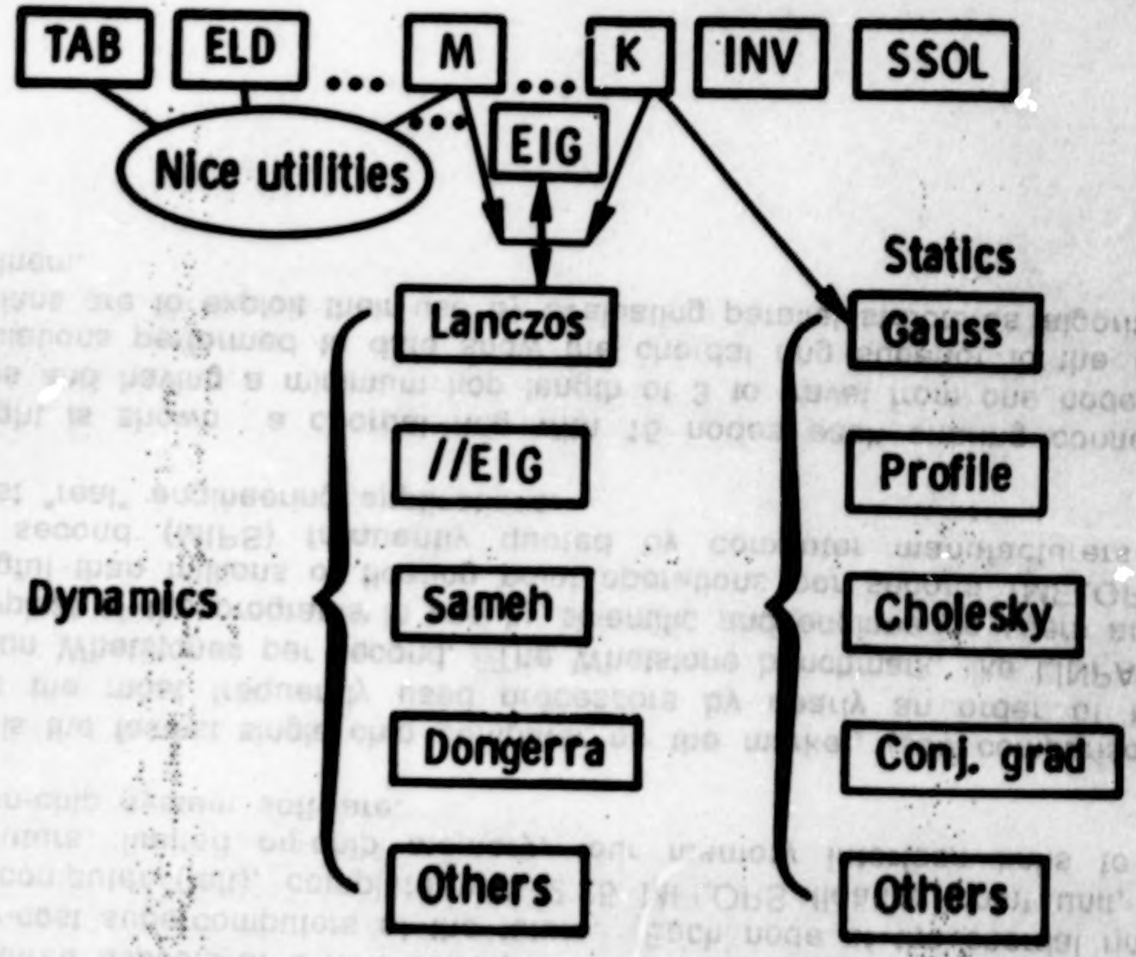
# MULTI-MICROPROCESSOR COMPUTER ARCHITECTURE

This slide shows three aspects of a new concept referred to as "chordal ring" for the design of a new family of low-cost supercomputers of the future. Each node of the chordal ring consists of an Inmos T800 computer (left), complete with 2.25 MFLOPS floating point unit, four links to companion computers, limited on-chip memory, four memory interface links to companion processors and on-chip system software.

The Inmos T800 is the fastest single chip computer on the market, (see comparison-lower right) and it surpasses the most frequently used processors by nearly an order of magnitude by performing 6 million Whetstones per second. The Whetstone benchmark, like LINPACK and other benchmarks, is typical of the programs in use by scientific and engineering users and is found to be more meaningful than millions of floating point operations per second (MFLOPS) or million instructions per second (MIPS) frequently quoted by computer manufacturers but of little credence for most "real" engineering applications.

On the upper right is shown a chordal ring with 15 nodes each sharing connections with 4 neighboring nodes and having a minimum hop length of 3 to travel from one node to any other node. The simulations performed to date show the chordal ring superior to the hypercubes in common use. Plans are to exploit their use by evaluating parallel structures algorithms from the CSM Testbed on them.

# PARALLEL STRUCTURAL ANALYSIS TESTBED



TAB ELD ... M ... K INV SSOL

... EIG

Nice utilities

Dynamics

Lanczos

//EIG

Sameh

Dongerra

Others

Statics

Gauss

Profile

Cholesky

Conj. grad

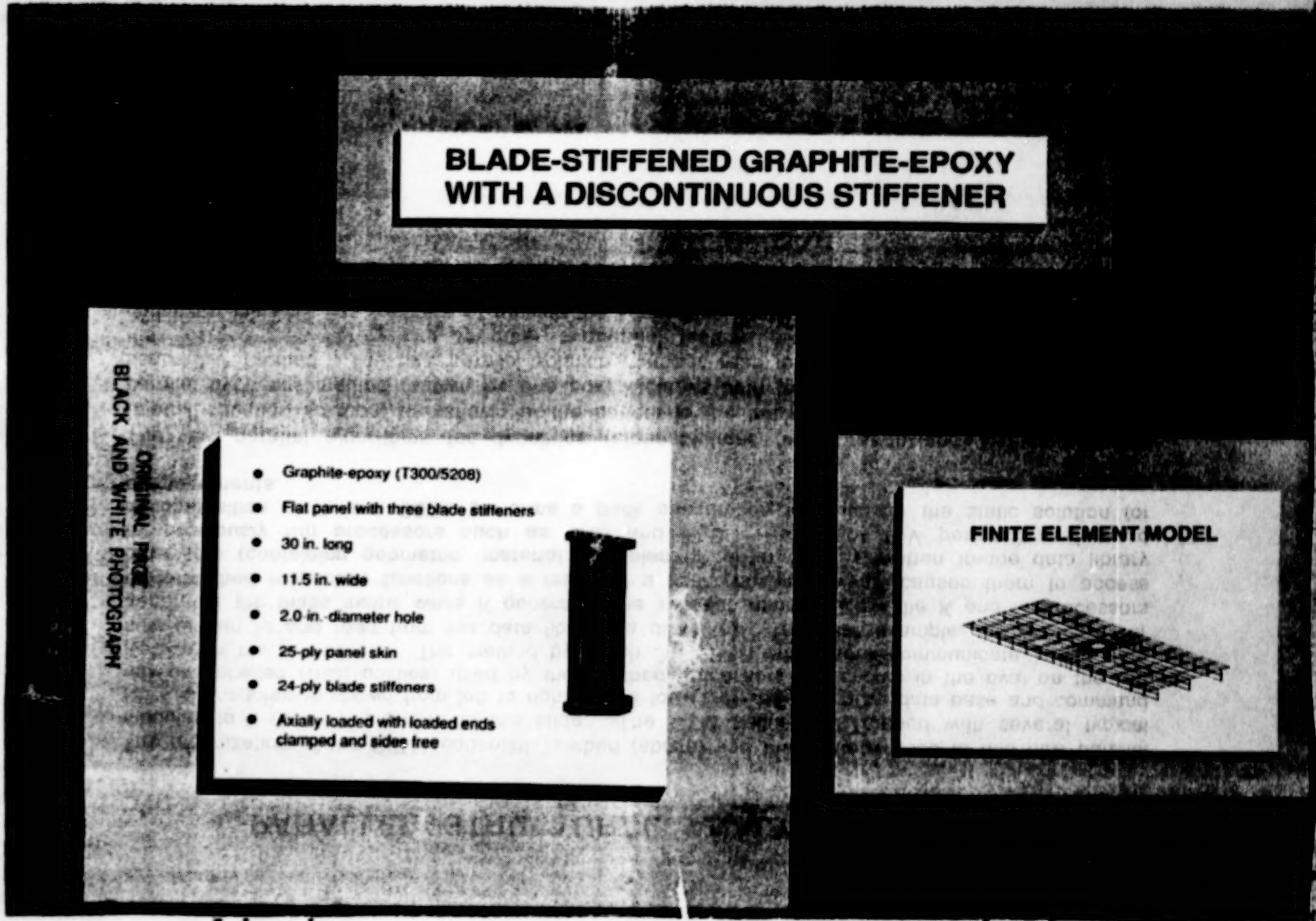
Others

NASA LANGLEY RESEARCH CENTER

# PARALLEL STRUCTURAL ANALYSIS TESTBED

The organization of the CSM sequential Testbed (above) and the strategy used to add new parallel modules to it (below) is shown in this slide. The CSM sequential Testbed with several typical (SPAR) modules is shown from left to right at the top (TAB...SSOL). The data base and command utilities (labelled NICE utilities) used by the Testbed processors are shown in the oval on the left just below the processors. The method by which the SPAR processors communicate is via data sets written to and read from the data library (a disk file). Thus, for example, the processor M generates the mass matrix while K generates the stiffness matrix. Both the K and M processors perform their respective functions as a result of a user command which causes them to access data sets (containing geometric, material and element data) already written in the data library by previously run processors such as TAB and ELD. Processor INV performs a forward decomposition of K, and SSOL performs a back substitution to calculate the static solution for displacements.

The new parallel algorithms for dynamics (eigensolutions for vibration analysis) and statics (matrix equation solution) are shown at the bottom to the left and right, respectively. Each new parallel dynamics method (shown by one box) replaces both the INV and EIG processors of the sequential Testbed. The new parallel solution methods read the K and M matrices directly from the Testbed data library just as other sequential testbed processors. Thus, on a parallel computer, the user has a choice of running the sequential algorithm used by the CSM Testbed or a parallel algorithm offering equivalent accuracy and a reduced computation time.



# BLADE-STIFFENED GRAPHITE-EPOXY WITH A DISCONTINUOUS STIFFENER

- Graphite-epoxy (T300/5208)
- Flat panel with three blade stiffeners
- 30 in. long
- 11.5 in. wide
- 2.0 in.-diameter hole
- 25-ply panel skin
- 24-ply blade stiffeners
- Axially loaded with loaded ends clamped and sides free

FINITE ELEMENT MODEL

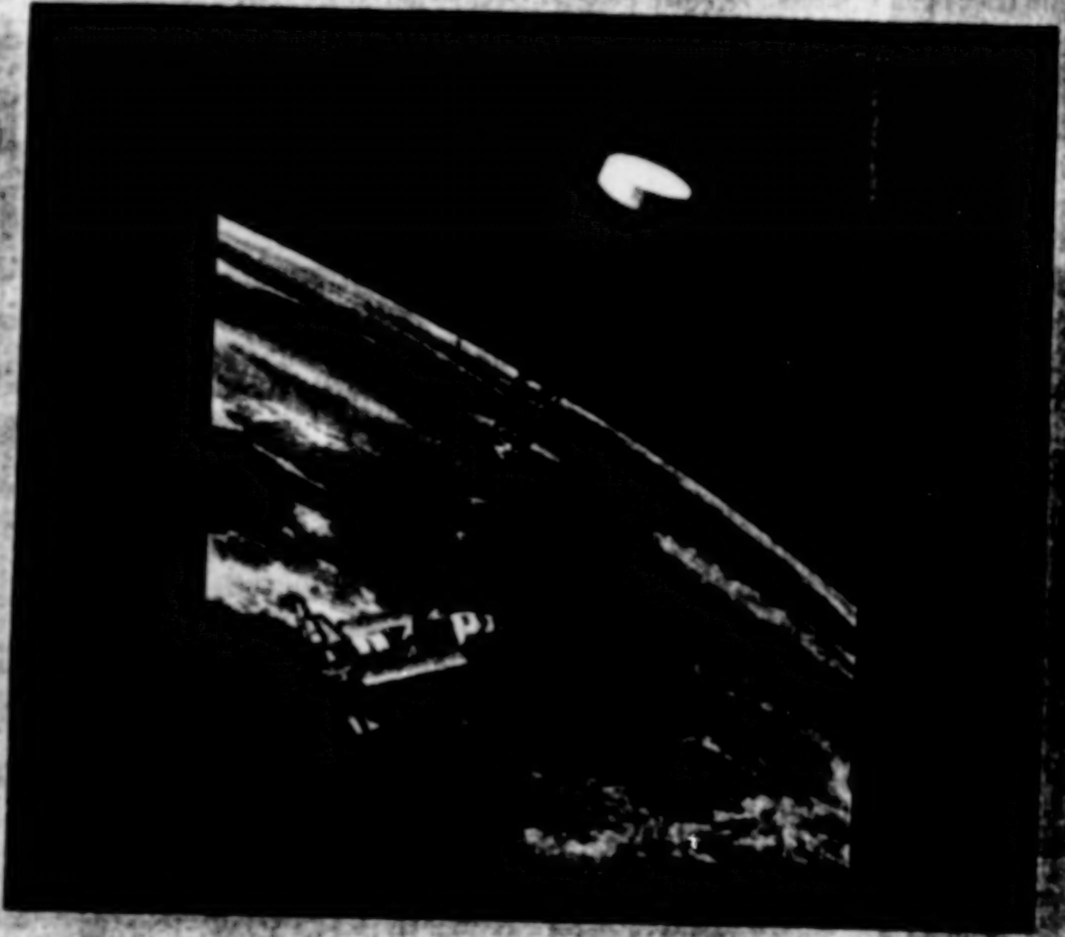
ORIGINAL PAGE
BLACK AND WHITE PHOTOGRAPH

# STIFFENED PANEL WITH CIRCULAR CUTOUT

A model of a 76.2 cm by 29.2 cm rectangular blade-stiffened aluminum panel with a 5.1 cm hole in the center is shown on this slide. It contains three 3.56 cm high stiffeners spaced 11.43 cm apart. The thicknesses of the plate and stiffeners are 0.254 cm. A more detailed description of the finite element model used (including input data) is contained in Appendix C of reference 10.

Three finite element models of this stiffened panel were used in this study: a coarse model (648 degrees-of-freedom), a medium-sized model (2328 degrees-of-freedom), and a detailed model (4392 degrees-of-freedom). The stiffness matrix for the coarse 108-node model of the panel contains 476 rows with a semi-bandwidth of 118, while that for the 2328 degree-of-freedom model has 1824 rows with a semi-bandwidth of 240. The behavior of these three stiffened panel models as well as a complementary space mast problem were used to evaluate the performance of the linear equation solvers and eigenvalue solvers developed for use on parallel computers.

# SPACE MAST PROBLEM



60 meter three-longeron truss
54 bays
165 nodes
486 degrees of freedom

ORIGINAL PAGE
BLACK AND WHITE PHOTOGRAPH

# SPACE MAST PROBLEM

The space Mast problem is based on a proposed space shuttle experiment to explore the dynamic characteristics of a 60-meter, 54-bay, 3-longeron deployable truss beam shown at the left. The finite element model contained 165 nodes and 486 degrees-of-freedom. Details of the model definition are contained in reference 8. The stiffness matrix for this model had a semi-bandwidth of only 18, considerably smaller than that for the stiffened panel.

A second, more detailed one-third length beam model of the space Mast with mid-point hinges, referred to as the Mini-Mast, was deployed and tested for static and dynamic characteristics at the NASA Langley Research Center. The Mini-Mast (ref. 9) consists of 18 bays containing thin graphite-epoxy tubular longerons, battens and diagonal members each with titanium tip connectors. The mass of the 111 titanium joints (0.7775 kg) is significant when compared to the light-weight tubular members. Thus the Mini-Mast is referred to as a "joint-dominated structure". The Mini-Mast is fixed at the three base points leaving 1980 of the total 1998 degrees-of-freedom active in the solution. Examination of the element interconnection pattern for joints reveals that the Mini-Mast has a small bandwidth (60) when compared to the panel problems (118 and 240).

ORIGINAL PAGE
BLACK AND WHITE PHOTOGRAPH

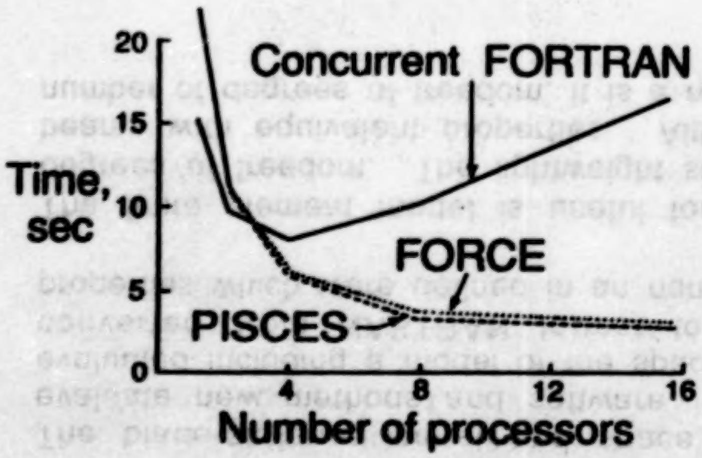
# SPACE STATION: POTENTIAL FOCUS PROBLEM

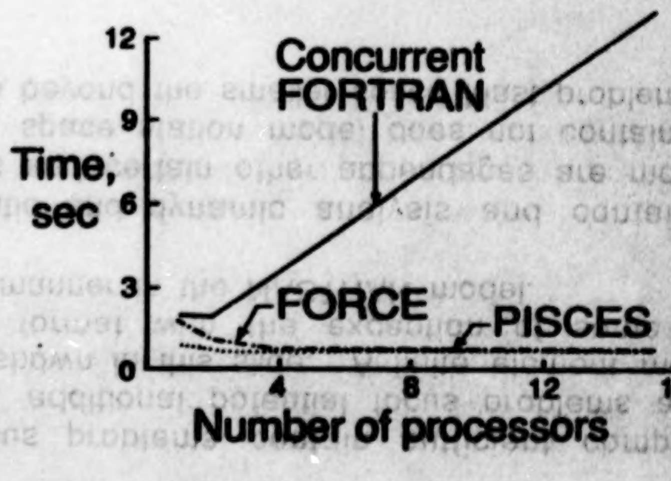
The blade-stiffened panel and space Mast focus problems contain sufficient complexity to evaluate new methods and software. However, additional potential focus problems are being evaluated including a model of the space station shown in this slide. A finite element model was converted from NASTRAN format to Testbed format with the exception of some material properties which were defined in an non-standard manner in the NASTRAN model.

The finite element model is useful for both static and dynamic analysis and contains 2328 degrees of freedom. The lightweight solar arrays and certain other appendages are modeled as beams with equivalent properties. Although the space station model does not contain a large number of degrees of freedom, it is a natural step beyond the smaller space Mast problems.

# PISCES AND FORCE REDUCE MATRIX EQUATION SOLUTION TIME

Problem: No parallel language standards or portability
*FORCE: FORTRAN extensions (U. of Colorado)
PISCES: Parallel programming environment with FORTRAN extensions (UVa)
Solve: 200 x 200 matrix with semi-bandwidth 50 (Duke, ICASE)


FACTORIZATION


BACK SUBSTITUTION

* Offers portability across FLEX, ENCORE, ALLIANT, HEP, SEQUENT

# PISCES AND FORCE REDUCE SOLUTION TIME

This slide describes an important way to achieve significant performance improvements on parallel computers. Until the development of Force and Pisces, there has been little accomplished to assist software developers on parallel computers to achieve efficient code while maintaining portability. In fact many feel significant efficiency must be sacrificed to achieve portability.

There is currently no standard for parallel language constructs and probably won't be until FORTRAN 9X is defined and agreed upon. In the meantime, Force was developed by Harry Jordan at the University of Colorado which maps a common parallel language (Force) into the differing parallel languages existing on different parallel computers. The method used the UNIX SED (stream editor) as a precompiler to convert the common parallel language to the corresponding parallel language (i.e. concurrent FORTRAN) supplied by each vendor for their computer.

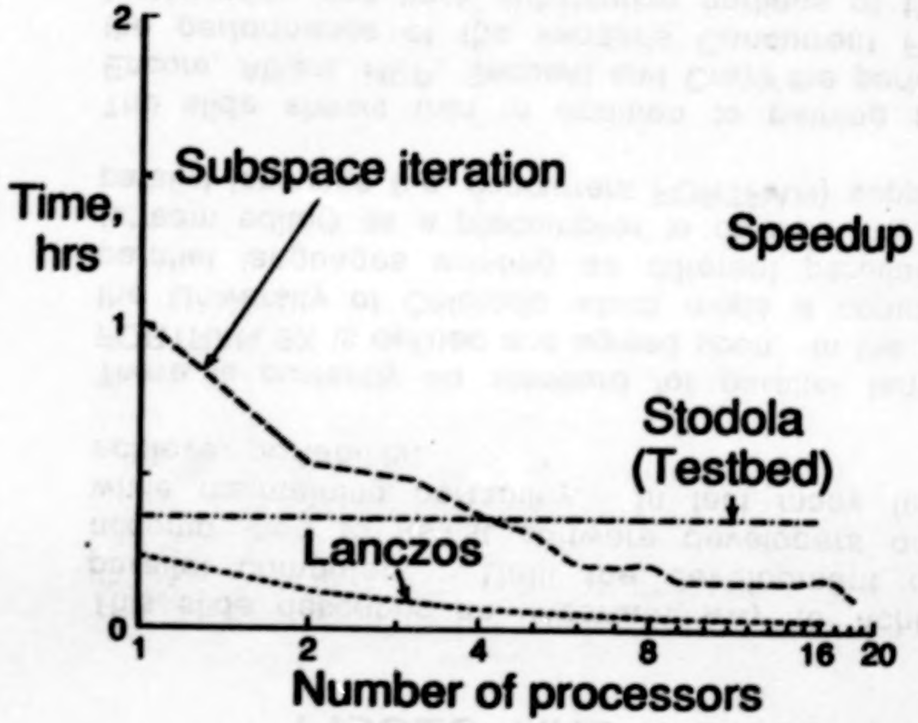
The slide shows that in addition to gaining portability across computers (including Flexible, Encore, Alliant, HEP, Sequent and Cray) the performance achieved by Force and Pisces exceeded the performance of the vendor's Concurrent FORTRAN by a significant amount for both the factorization and back substitution portions of the matrix equation solution. The trends actually degrade seriously for concurrent FORTRAN for beyond just a few processors, while the situation continues to improve for Force and Pisces.

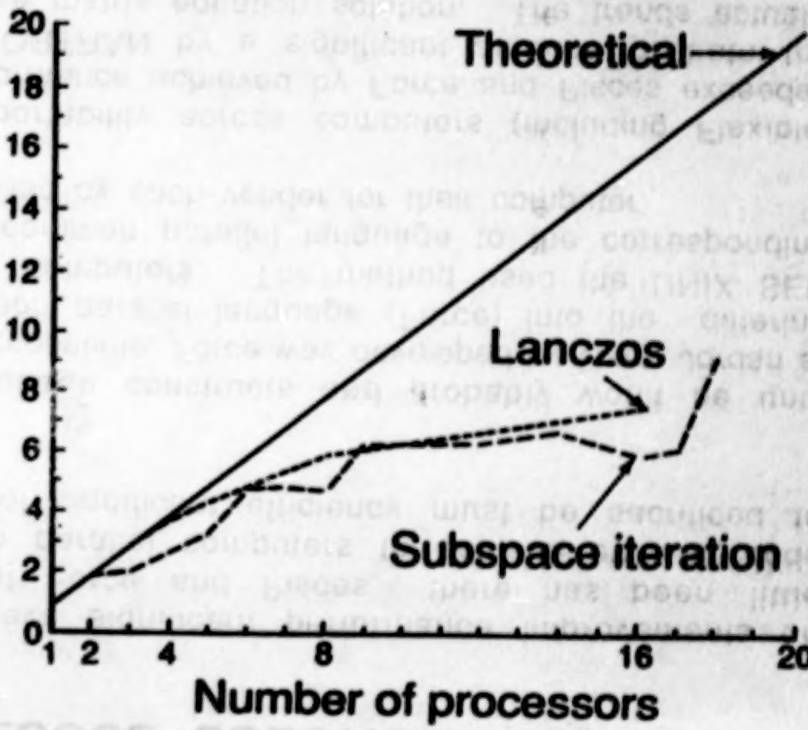
Code written in Force requires no changes to run on another parallel computer on which Force is running. Thus, new solution methods developed on the Flexible/32 research parallel computer can be transferred to and run on the Cray 2 without changes. All those developing new parallel algorithms for CSM are encouraged to use Force for both ease and commonality of coding and portability to other parallel computers.

# PARALLEL EIGENSOLVERS REDUCE SOLUTION TIME

Solve: $Kx = \lambda Mx$

Example: 10 lowest vibration frequencies of graphite-epoxy panel by Lanczos (in-house) and sub-space iteration (Duke University)





# PARALLEL EIGENSOLVERS REDUCE SOLUTION TIME

The determination of the fundamental (lowest) natural vibration frequencies and associated mode shapes is a key step used to uncover and correct potential failures or problem areas in most complex structures. However, the computation time taken by finite element codes to evaluate these natural frequencies is significant, often the most computationally intensive part of structural analysis calculations. There is a continuing need to reduce this computation time. This slide shows significant reductions in computation time achieved by two parallel eigensolution methods. The objective of both the Lanczos and Subspace Iteration method is to solve the eigenvalue problem $Kx = xMx$ for the ten lowest vibration frequencies for the stiffened panel CSM focus problem.

The plot on the left shows the reduction in computation time achieved by the Lanczos method and the subspace iteration as the number of processors increases. Despite the slightly greater reduction achieved by the subspace iteration method, the Lanczos method took less time overall than the subspace iteration method and the sequential Testbed solver (horizontal line) regardless of the number of processors.

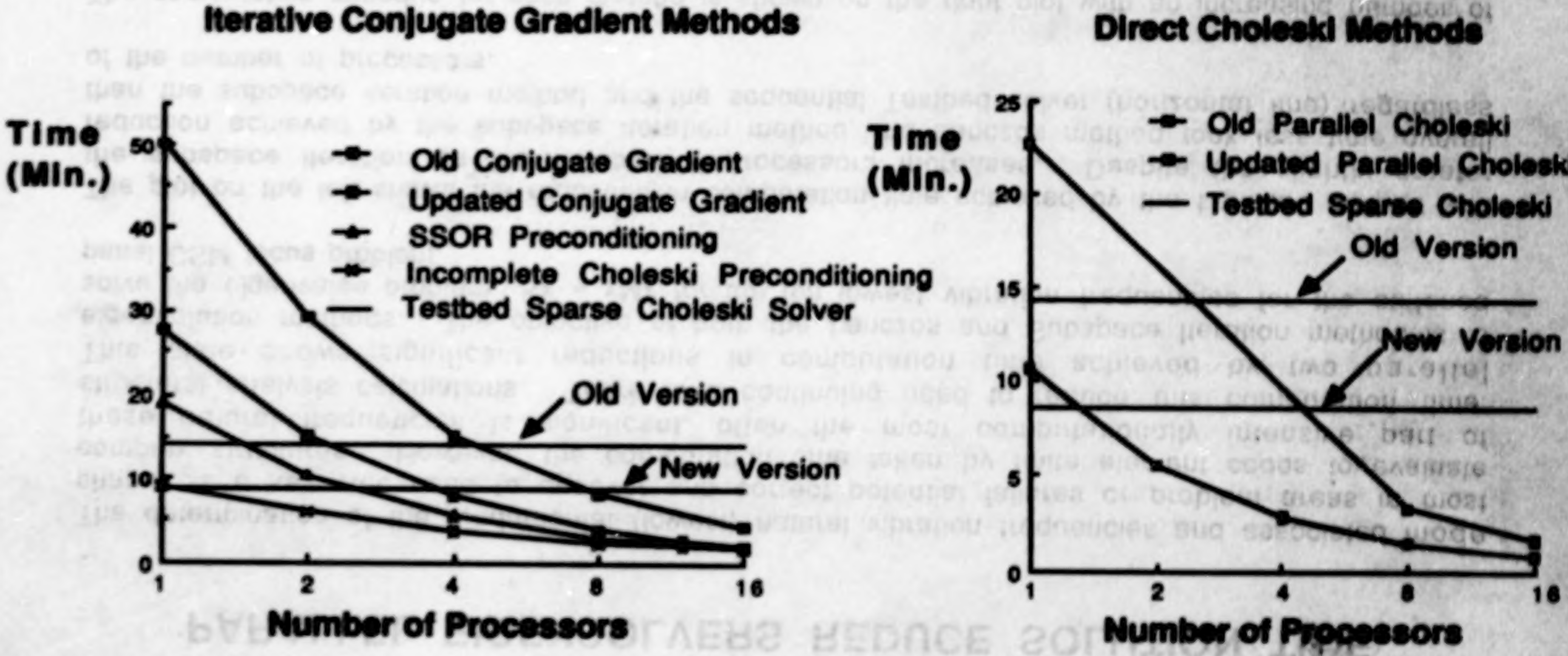
The computation speedup for each method is shown on the right plot with an increasing number of processors along the abscissa. Both methods fall below the theoretical limit (maximum speedup) which indicates further reductions in computation time may be possible by introducing further refinements to the parallel methods.

Further details of these results can be found in reference 2.

# Preconditioners and Refined Code Improve Performance of Equation Solvers

Solve Ku = f

Comparison of Execution Times for Panel Example Problem
with 1824 Equations



Iterative Conjugate Gradient Methods

Direct Choleski Methods

# PARALLEL EQUATION SOLVERS REDUCE SOLUTION TIME

The solution of the system of load-displacement equations, $K u = f$ is often the most time consuming portion of the solution of linear finite element structural analysis problems. As the size of the stiffness matrix, K, increases, the proportion of time spent in equation solution increases at a superlinear rate, approaching in excess of 90 percent of the time spent for larger structural analysis problems. Reductions achieved in computation of the structural displacements have the direct benefit of permitting the solution of larger more complex problems, without resorting to the simplifying assumptions used by many to bypass large computation times for the static structural analysis.

Results obtained for two methods, Conjugate gradient and Choleski are shown in the plot at the lower left. For this 1824 degree-of-freedom stiffened panel problem, the best times obtained by iterative (left) and direct (right) solvers for one processor are competitive with the sparse Choleski solver in the new version of the Testbed.. More significant is that all parallel solvers developed eventually are faster than the new Testbed solver, with the Incomplete Choleski Preconditioned Conjugate Gradient method performing the best for the iterative methods and the Updated Parallel Choleski performing best for the direct methods. These results are typical in that smaller problems with smaller bandwidths may not perform as well in parallel while larger problems with the same or increased bandwidths perform even better.

# PARALLEL STRUCTURAL METHODS: FUTURE

- Solve Nonlinear & Buckling Problems
- Parallel Matrix Generation (Element, Stiffness & Mass)
- Parallel Substructuring Methods
- Portability across Parallel Computers

# PARALLEL STRUCTURAL METHODS: FUTURE

Our future work in parallel structural methods includes many facets ranging from parallel and vector structural analysis methods on the Cray 2 to development of a new architecture (chordal ring) 15-processor parallel computer based on high-speed transputer processors. However, the four items shown on this slide represent major directions the parallel structural methods group plans for the future.

Based on the success achieved for the parallel solution of linear static and vibration analysis problems, we plan to extend our methods to include nonlinear analysis and buckling problems. The nonlinear analysis consists of repeated linear analyses to exploit our new efficient linear solvers controlled by an arc-length (or other) search methods. The buckling analysis algorithm is quite similar to the vibration analysis methods (Lanczos and Subspace Iteration) where the Kg matrix is used in place of the mass matrix.

A second direction is to extend the benefits of parallel solution to the generation of elemental and global stiffness matrices and mass matrices. Although major computation time reductions are not expected for this, it is a step towards total parallelism and achieving improved efficiency. Providing a parallel substructuring capability in which different substructures can be generated in parallel and the solution obtained either via the parallel solvers or a new parallel substructuring solver is a challenging objective. Plans are to add such parallel substructuring capability to the CSM testbed, based on the primitive "hooks" in the testbed used for modal synthesis.

The final item is to demonstrate portability of typical testbed processors across several parallel computers. Work is currently underway aimed at achieving this goal by using the Force extensions to concurrent Fortran (ref. 7).

# REFERENCES

1. Storaasli, O., Poole, E., Ortega, J., Cleary, A.and Vaughan, C., "Solution of Structural Analysis Problems on a Parallel Computer," AIAA Paper 88-2287-CP, Proceedings of the 29th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference, Williamsburg, Virginia, April 18-20,1988.

2. Storaasli, O., Bostic, S., Patrick, M., Mahajan, U. and Ma, S., "Three Parallel Computation Methods for Structural Vibration Analysis," AIAA Paper 88-2391-CP, Proceedings of the 29th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference, Williamsburg, Virginia, April 18-20, 1988.

3. Cleary, A., Harrar, D. and Ortega, J., "Gaussian Elimination and Choleski Factorization on the FLEX/32,", University of Virginia Applied Mathematics Report RM-86-13, 1986.

4. Bostic, S. and Fulton, R., "A Lanczos Eigenvalue Method on a Parallel Computer", AIAA Paper 87-0725-CP, Proceedings of the 28th AIAA/ASME/ASCE/AHS Structures, Structural Dynamics and Materials Conference, Monterey, California, April 6-8, 1987.

5. Storaasli, O., Ransom, J. and Fulton, R., "Structural Dynamic Analysis on a Parallel Computer: The Finite Element Machine," Computers and Structures, Vol. 26, No. 4, pp. 551-559, 1987.

6. Storaasli, O., Peebles, S., Knott, J., Crockett, T. and Adams, L., "The Finite Element Machine: An Experiment in Parallel Processing. NASA TM 84514, July, 1982, also in Research in Structural and Solid Mechanics, NASA CP 2245, pp. 201-217, 1982.

7. Jordan, H., Benten, M., Arenstorf, N. and Ramanan, A., "Force User's Manual," Department of Electrical and Computer Engineering Technical Report 86-1-4R, University of Colorado, Boulder, CO, October, 1987.

8. Horta, L., Walsh, J., Horner, G. and Bailey, J., "Analysis and Simulation of the MAST (COFS-1 Flight Hardware)," NASA CP 2447, Part 1, Nov. 18-21, 1986, pp. 515-532.

9. Adams, L., "Design, Development and Fabrication of a Deployable/Retractable Truss Beam Model for Large Space Structures Application," NASA CR 178287, June 1987.

10. Lotts, T. et al., "Introduction to the CSM Testbed: NICE/SPAR," NASA TM-89096, 1986.

N89
29776

UNCLAS

AEROSPACE TECHNOLOGY DIRECTORATE

STRUCTURES DIVISION
Structural Dynamics Branch

NASA
Lewis Research Center

ND315753 N89-29776

S3-39
148
511944
P-14

# COMPUTATIONAL STRUCTURAL METHODS AT NASA LEWIS

NOVEMBER 18, 1987
L J KIRALY

PRECEDING PAGE BLANK NOT FILMED

PAGE 60 INTENTIONALLY BLANK

# COMPUTATIONAL STRUCTURAL METHODS

## SOME HISTORY:

*Custom Architecture Parallel Processing System - CAPPS*
*Simulation of wind turbine dynamics & controls.*

*Real Time Multi - Processor System - RTMPS*
*Simulation of engine systems control dynamics.*

*Computational Structural Methods Activity - initiated 2 years ago - to complement on-going computational structural analysis methods development.*

The Structural Dynamics Branch at Lewis conducts research in propulsion and power systems, and in mechanical systems applications. We have four major areas of work which are:

- Aeroelasticity
  - Classical (computational)
  - Computational (time domain)
  - Experimental
  - Applications (Turboprop, turbofan, turbopump, and advanced core technology)
- Vibration Control
  - Active methods
  - Passive methods
  - Forcing functions
  - Applications (Electromagnetic dampers, magnetic bearings, cryo turbomachinery)
- Dynamic Systems
  - Micro-gravity robotics systems
  - Parameter identification
  - Applications (Space lab, SP100 Engine, NASP seals, tethered satellites)
- Computational Methods
  - Algorithms for modern computing
  - Engineering data analysis
  - Parallel architecture computers
  - Applications (Parallel FE methods, transputer lab, transients analysis)

Our computational methods activity relates to other branch and lab programs. It is based on current needs, past activities, and is coordinated with activities in the fluid mechanics division and the computer services division.

# COMPUTATIONAL STRUCTURAL METHODS

## PROGRAM OBJECTIVE:

*EXPLOIT MODERN COMPUTER HARDWARE & SOFTWARE TO FUNDAMENTALLY IMPROVE THE USE OF COMPUTERS FOR SOLVING STRUCTURAL PROBLEMS.*

## WORK ELEMENTS:

- *ALGORITHMS FOR MODERN COMPUTING*
- *ENGINEERING DATA ANALYSIS*
- *PARALLEL ARCHITECTURE COMPUTERS*
- *APPLICATION STUDIES*

The goal of our work is to exploit modern computing architectures. It is a new activity for Lewis. Our initial work has been directed to more fundamental concerns dealing with how we might formulate new algorithms to take advantage of parallel computing and how these new computers might be applied to data-taking and analysis. Our longer-term goal is to make new methods part of design and analysis practice with the engine simulator activity also underway at Lewis.

# COMPUTATIONAL STRUCTURAL METHODS

## KEY THRUSTS:

### *INNOVATIVE METHODS*

- ***High performance potential***
- ***Applications which require parallel computing***
- ***Applications which greatly benefit from parallel computing***

### *METHODS FOR ADAPTING EXISTING CODES*

- ***FORTRAN conversion***
- ***Finite Element Modeling***

### *REQUISITE SOFTWARE TOOLS*

- ***Code analysis***
- ***Architecture Evaluation***
- ***Architecture Synthesis***

We have placed strong emphasis on new innovative approaches which we feel will offer significant performance advantages in future structures codes. Along with this activity we have started to identify methods which may be employed to successfully re-utilize the large stock of existing codes that we currently use, because of the tremendous investment that the agency has in these codes. Finally, it became apparent that some effort was also required in developing and updating software tools to analyze the performance potential of alternative methods on alternative architecture processors.

# COMPUTATIONAL STRUCTURAL METHODS

## APPROACH:

*FOCUS ON FUNDAMENTALS - LEADING TO APPLICATIONS*

*FORM LAB-WIDE CSM RESEARCH TEAM*

*REPRESENTATIVE PROBLEM CASE STUDY*

*INSTALL RE-CONFIGURABLE ARCHITECTURE TRANSPUTER*

*ESTABLISH USER COMMUNITY*

*"EVENTUALLY" - PURCHASE COMMERCIAL SYSTEM*

Our approach starts with fundamentals. There are many interested parties at Lewis who have come together to form a lab-wide team. We currently have representatives from Structures, Computational Fluid Mechanics, ICOMP, and Computer Services Division. By pooling our resources and studying representative problems we hope to develop some common understanding which will lead to a user community at Lewis. We will be attempting to pool resources to procure a commercial parallel processing computer to complement the research architecture processors currently on hand (the transputer and the hypercluster architectures).

# COMPUTATIONAL STRUCTURAL METHODS

## RESOURCES:

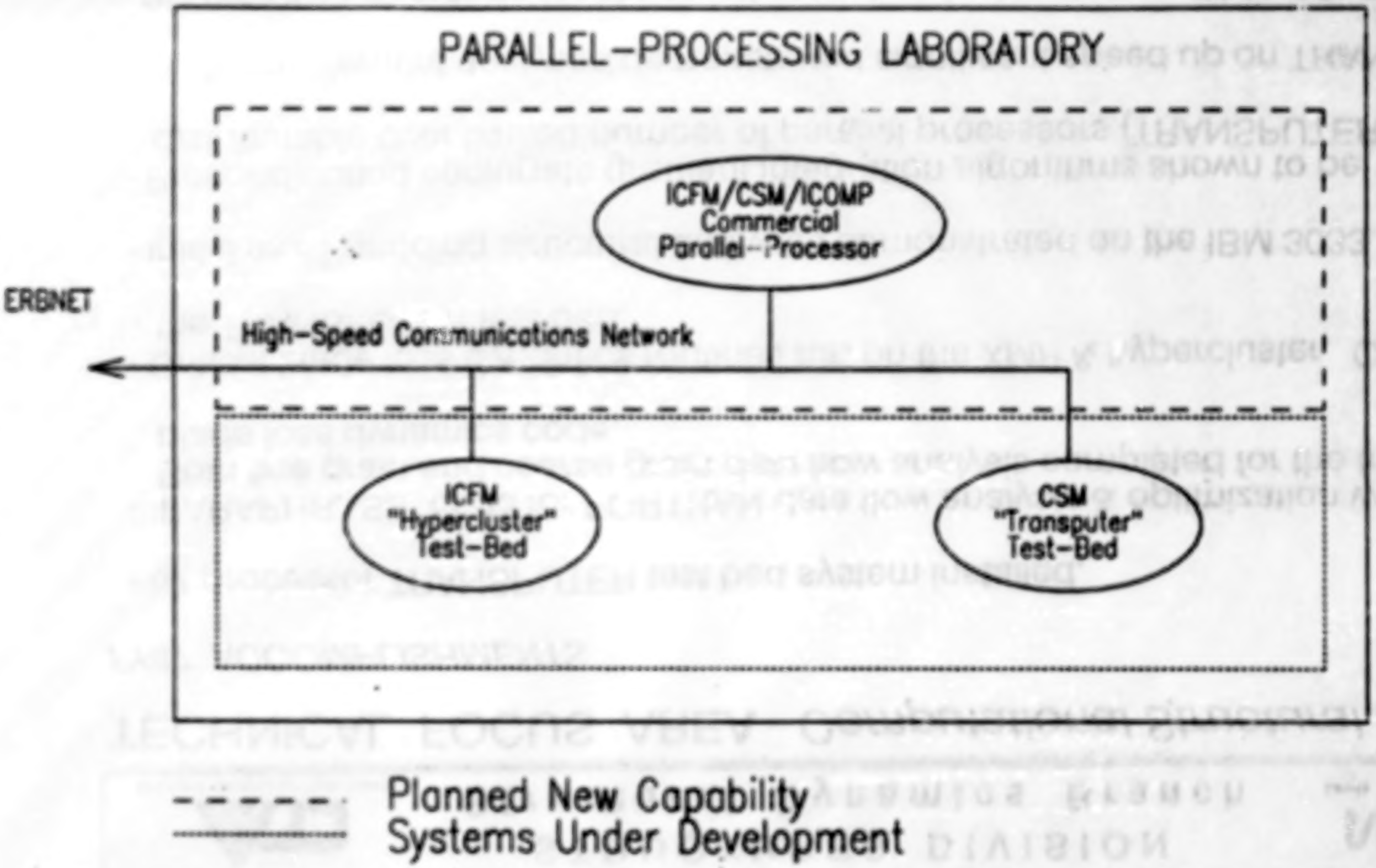
*IN-HOUSE CSD, ICFM, ICOMP & CSM TEAM*

*67 PROCESSOR TRANPUSTER ARRAY TEST BED*

*NETWORKING RESOURCES: ERBNET, NASNET*

*OTHERS COMPUTERS: HYPERCLUSTER, XMP, CRAY II*

*PROPOSED COMMERCIAL SYSTEM*


PARALLEL-PROCESSING LABORATORY
ICFM/CSM/ICOMP Commercial Parallel-Processor
High-Speed Communications Network
ERBNET
ICFM "Hypercluster" Test-Bed
CSM "Transputer" Test-Bed
Planned New Capability
Systems Under Development

# TECHNICAL FOCUS AREA - *Computational Structural Methods*

## *FY87 ACCOMPLISHMENTS*

- 67 processor TRANSPUTER test bed system installed.
- 'PARAPHRASE' code for FORTRAN data flow analysis & optimization was installed. Both fine grain and coarse grain data flow analysis completed for the transient blade loss dynamics code.
- Critical blade loss dynamics routines run on the XMP & hypercluster. Coded for the TRANSPUTER test bed.
- Initial multi-gridding structural analyses demonstrated on the IBM 3033.
- Preconditioned conjugate gradient integration algorithms shown to be distributable over limited number of parallel processors (TRANSPUTERS).
- 2D Finite element analyses demonstrated significant speed up on TRANSPUTERS.
- 2D graphics primitives for structural modeling/animation on TRANSPUTER.
- A general model of parallel processors (as seen by structures codes) using both deterministic and statistical factors formulated for algorithm assessment.
- Space station power systems control strategies were simulated on the CAPPS.

## TECHNICAL FOCUS AREA - *Computational Structural Methods*

*FY88 PLANS*

- Demonstrate structural multi-gridding analyses on the TRANSPUTER array.
- Formulate parallel algorithms for real-time (LQR) rotor response control, and develop real-time rotor response simulation codes on the TRANSPUTER test bed.
- Demonstrate a general architecture assessment model for structures codes.
- Demonstrate binary-tree sub-domain decomposition frontal method eigen-solver codes.
- Demonstrate a TRANSPUTER library of 'GKS-style' graphics primitives for animation.
- Formulate TRANSPUTER 3D FE analyses with out-of-core solution strategies.
- Formulate 2D FE re-meshing code to optimally re-distribute and balance the computing load to an array of TRANSPUTERS.
- Assess processing array limits for preconditioned conjugate gradient integration.
- Compute the aerodynamic coefficients across the surface of an ATP blade in parallel.
- Use 'PARAPHRASE' to optimally convert existing FORTRAN codes to OCCAM.

**STRUCTURES DIVISION**
**Structural Dynamics Branch**

# COMPUTATIONAL STRUCTURAL METHODS
## *SUMMARY OF CURRENT & PLANNED ACTIVITIES*

| | NOVEL METHODS | EXISTING CODE USE | PAR. TOOLS/ DEMO's |
|---|---|---|---|
| **ALGORITHMS** | *- Structural Multi-gridding*<br>*- Finite Time Dynamics FE* | *- FORTRAN to OCCAM Conversion* | *- Real Time Adaptive Rotor Control* |
| **DATA ANALYSIS** | *- Digital Comb Filter* | | *- 'EASY-FLO' Course Grain data Flow* |
| **PARALLEL COMPUTING** | *- Pre-cond Conj-Grad Integration*<br>*- Sub-Domain Eigen-solver methods* | *- PARAPHRASE Evaluation*<br>*- ATP Blade Aero Coefficients* | *- TRANSPUTER primitives*<br>*- Architecture Modeling*<br>*- Architecture Synthesis* |
| **APPLICATIONS** | | *- Blade Transient Architecture Assessment* | *- TRANSPUTER Graphics Engine*<br>*- 2D FE model re-meshing*<br>*- TRANSPUTER FE Work-station* |

N89
29777
UNCLAS

N89-29777

SV158045 54-39
29P.
211945
P-31

**SPARTA, INC.**

# TRANSPUTER FINITE ELEMENT SOLVER

See p 77

SPARTA, Inc.
4901 Corporate Drive
Huntsville, AL 35805
(205) 837-5200

1187-005/C

ORIGINAL PAGE
BLACK AND WHITE PHOTOGRAPH

ORIGINAL PAGE
BLACK AND WHITE PHOTOGRAPH

SPARTA, INC.

# INTRODUCTION

- SPARTA SBIR AWARD
- GENERAL TRANSPUTER INFORMATION
- RESULTS OF FEASIBILITY STUDY
- PROSPECTS FOR A LARGE-SCALE TFES

N89-29772

## Introduction--Transputer Based Finite Element Solver

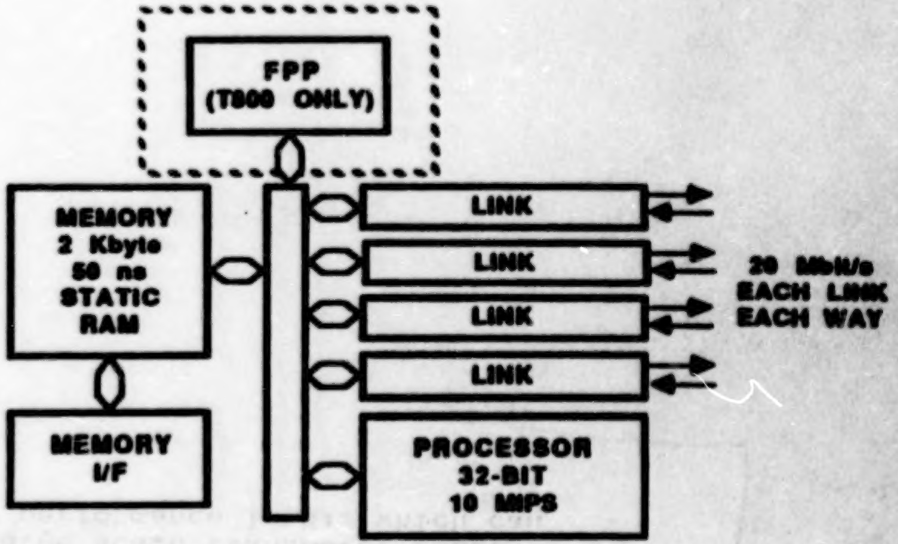
In January 1987, SPARTA received a Phase I SBIR award from the NASA Lewis Research Center to investigate the feasibility of a finite element solver implemented on multiple VLSI processors. The transputer was chosen as the processor for the feasibility study since it combined low cost with high performance and was specifically designed to directly link with other transputers to form networks of multiple processors. This presentation consists of three parts:

(1) a brief description of transputers,
(2) a summary of the SBIR feasibility study, and
(3) a discussion of issues concerning a large scale transputer based finite element solver (TBFES) and the performance levels which can be expected.

# GENERAL INFORMATION - INMOS T414 AND T800 TRANSPUTERS

SPARTA, INC.

- T414/T800 SPECIFICATIONS
  - 10 MIPS
  - 4 BIDIRECTIONAL LINKS @ 20 Mbits/sec
  - 4 Gbyte ADDRESS SPACE
  - 1.2 MICRON CIRCUITRY
  - TRANSPUTER CHIP OCCUPIES ONLY 2 SQUARE INCHES
- T800 FPP CAN SUSTAIN 1.5 Mflops
- PROGRAMMING LANGUAGES
  - OCCAM
  - PASCAL
  - C
  - FORTRAN
  - FORTH


FPP
(T800 ONLY)
MEMORY
2 Kbyte
50 ns
STATIC
RAM
MEMORY
I/F
LINK
LINK
LINK
LINK
PROCESSOR
32-BIT
10 MIPS
20 Mbit/s
EACH LINK
EACH WAY

TRANSPUTER CHIP CONFIGURATION

## Transputer General Information

### Specifications

The transputer, developed by INMOS Corporation, is a complete VLSI computer on a single chip. Two 32-bit versions exist: the T414, which has an integer processor, and the T800, which has a floating point processor. Transputers have 10 MIP processors, 2 (T414) or 4 (T800) Kbytes of static cache memory, interfaces to external RAM through a 32 bit address bus (to access up to 4 Gbytes of memory), and four bidirectional serial ports, called links, to directly communicate with other transputers. Links are capable of transmitting data a rate of 20 Mbits/sec. Its 1.2 micron circuitry enables a transputer chip to occupy less than two square inches on a printed circuit board; four transputers with 1 Mbyte of RAM each can fit on a single IBM PC plug-in card. The T800 is capable of sustaining 1.5 million floating point operations per second (Mflops), so a plug-in card with four T800's has 6 Mflops--about 18 times the sustained floating point performance of a VAX 11/780.

### Languages

The transputer architecture was designed to implement occam, a high-level structured language for parallel processing. Occam is strongly typed and has a number of built-in constructs useful for defining parallel tasks, for communicating between parallel tasks (on the same transputer or on other transputers connected to it through links), and for real-time control of program execution. Since occam maps closely to the transputer architecture, programs written in occam are very efficient and take full advantage of the transputer's unique capabilities. A few desirable features have not yet been implemented, however. The current release of occam (occam2) does not have structured variables or pointers, does not allow recursion, and does not support functions. In addition, the lack of a transputer operating system and imperfect error locators makes debugging very challenging.

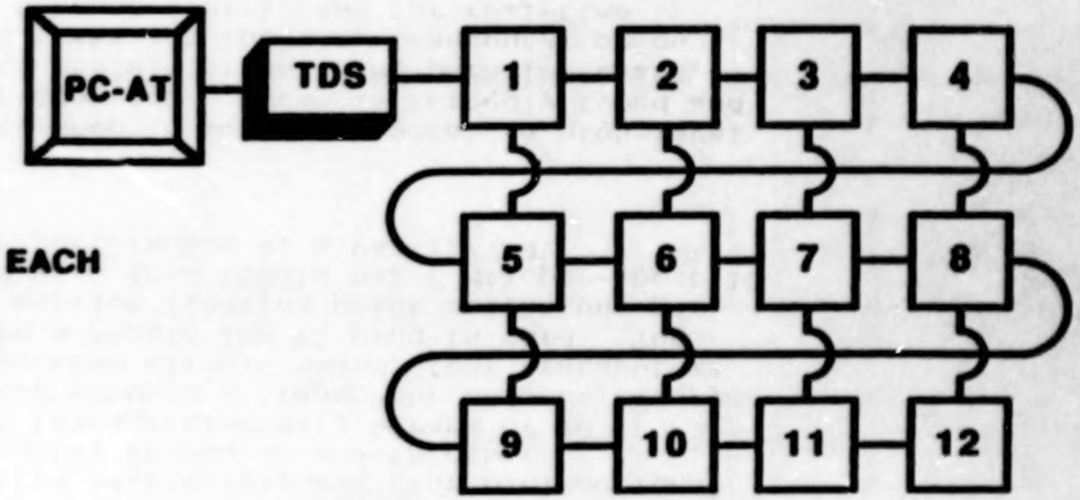
Pascal, C, FORTRAN and Forth compilers are also available.

# HARDWARE FOR SBIR FEASIBILITY STUDY

SPARTA, INC.

**HARDWARE:**
- IBM PC-AT CLONE
- ONE T414A TDS WITH 2 MBytes
- TWELVE T414Bs WITH 256 KBytes EACH
- TOTAL SYSTEM COST OF $15,000

**A TRANSPUTER BASED FINITE ELEMENT SOLVER**



TRANSPUTER NETWORK
ARRANGED IN A HELICAL GRID

## Hardware

The hardware used for the feasibility study included an IBM PC-AT compatible computer, an INMOS Transputer Development System (TDS) board with a T414 transputer and 2 Mbytes of RAM, and twelve T414's with 256 Kbytes of RAM each. The AT compatible served as the host computer by providing keyboard, screen and disk access to and from the TDS; the PC microprocessor and RAM memory are not used for any transputer operation. The thirteen transputers were configured in a helical grid, a topology which has relatively short communication path lengths.

Towards the end of the study, SPARTA received a pre-production prototype T800 for testing and evaluation of its floating point processor. This T800 was a revision "A" and ran at only 0.6 Mflops, less than half of the current production T800 (revision "C") capability. Sequential versions of the parallel finite element code (derived by eliminating task distribution) was run on both the T800A and the TDS's T414A transputers for comparison with the parallel code running on the entire network of T414's.

The minicomputers used for comparison were an Apollo DN660 and a VAX 11/780 with a floating point accelerator (FPA). The finite element code on these machines, although written in FORTRAN, is a direct translation of the occam code running on the transputer FE solvers (neglecting communication and parallel logic code) and executes the same sequence of steps to arrive at a solution. Timing results were obtained when both minicomputers had no other loads so the reported CPU times were equal to the actual elapsed times.

# SBIR FEASIBILITY DEMONSTRATION

SPARTA, INC.

- GOALS
  - INVESTIGATE WAYS OF DECOMPOSING FE METHOD ON A NETWORK OF TRANSPUTERS
  - IMPLEMENT A TBFES
  - COMPARE COST AND PERFORMANCE TO EXISTING COMPUTER SYSTEMS

PAGE 93 INTENTIONALLY BLANK

SPARTA, INC.

# FE SOLUTION METHODS

- DIRECT METHOD
  - ASSEMBLY OF GLOBAL STIFFNESS MATRIX
    - HIGHLY PARALLEL
    - COMPUTE BOUND
  - SOLUTION OF LINEAR EQUATIONS WITH GAUSSIAN ELIMINATION
    - GAUSS-JORDAN ELIMINATION HAS EFFICIENT PARALLEL IMPLEMENTATION
    - COMPUTE BOUND
- FRONTAL METHOD
  - LOWER OPERATIONS COUNT
  - MULTIPLE CONCURRENT FRONTS

PRECEDING PAGE BLANK NOT FILMED

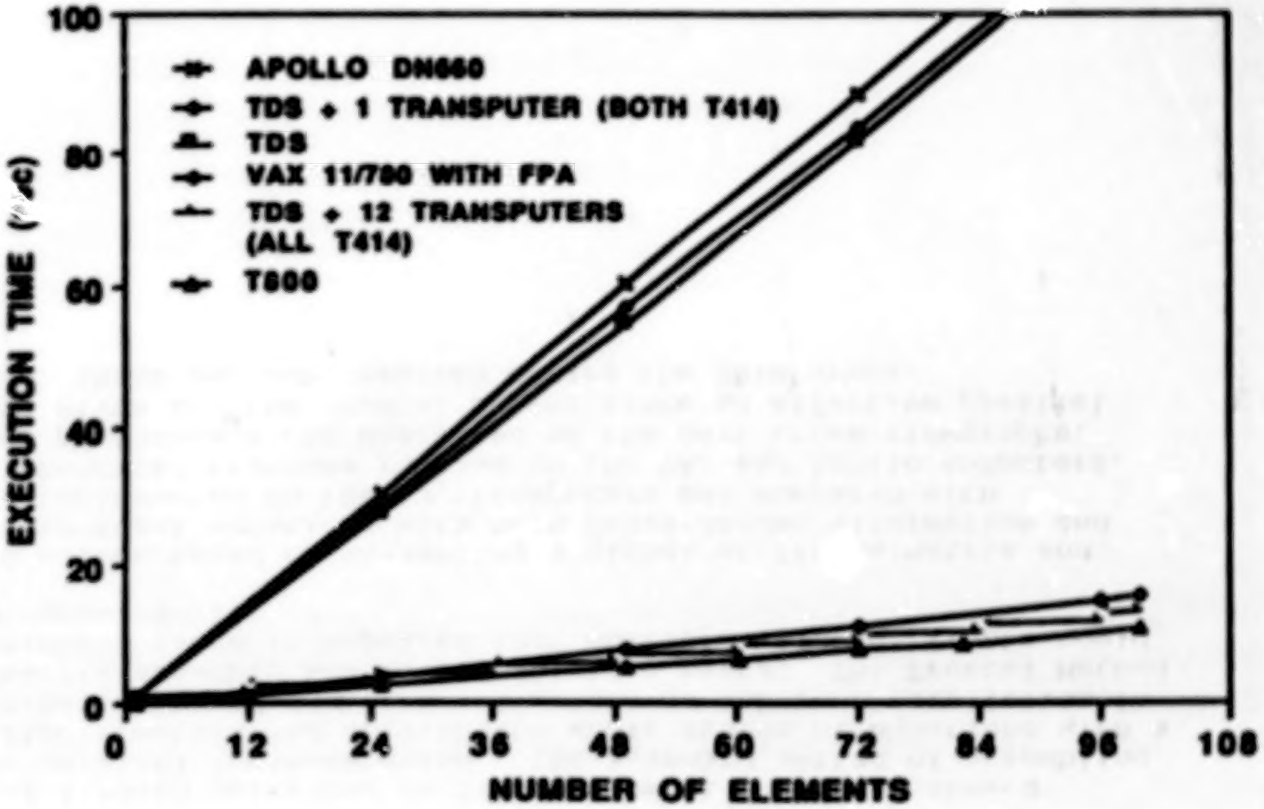
## Solution Methods

Two methods of finding solutions to finite element equations showed potential for parallel implementation. The standard method of assembling a global stiffness matrix and solving the whole system of equations with a Gaussian elimination-based equation solver showed inherent parallelism in both global matrix assembly and matrix solution parts. The Frontal Method was also considered since it appeared that multiple solution fronts could be initiated concurrently.

A complete FE solver based on assembling a global stiffness matrix and solving the structural equations with both Gauss-Jordan elimination and inversion was implemented on the 13 transputers and compared with functionally identical programs running on the VAX and Apollo computers. Its design and performance are described on the next three viewgraphs. No results are given for the frontal solver since an effective parallel implementation scheme was not resolved during the SBIR study.

# EXECUTION TIMES FOR GLOBAL STIFFNESS MATRIX ASSEMBLY

SPARTA, INC.



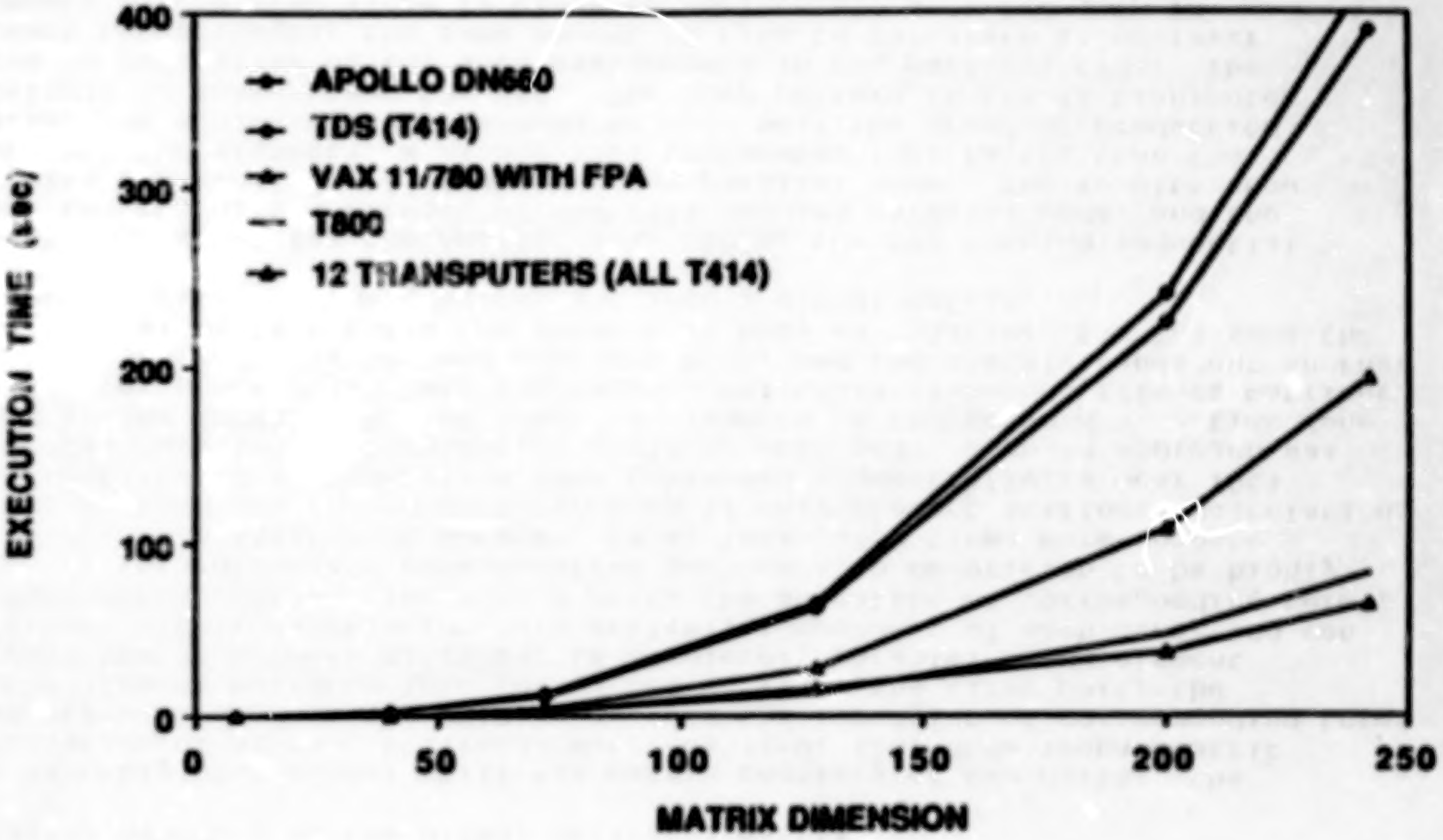
## Parallel Assembly of the Global Stiffness Matrix

The assembly of a global stiffness matrix consists of two parts: the calculation of element stiffness matrices (2-D, four node isoparametric elements were used in the SBIR study) and the summation of corresponding terms of the element matrices into one global matrix. The first part, the calculation of element matrices, is completely parallel since element stiffness matrix calculations are entirely idependent of each other and can be done concurrently. The second part, the summation of corresponding terms, requires interprocessor communication but can also be written to be highly parallel. The first part however, is at least 1000 times more compute intensive than the second part, so even if only element stiffness calculations were distributed and summation were performed sequentially, a near 100% efficient parallel implementation could be obtained. Such an approach was taken in the TBFES: the TDS sends 12 elements to the network at a time (one element per processor), lets the network calculate element stiffness matrices, waits for the network to send matrices back, and immediately sends out another set of 12 elements. While the network is busy calculating, the TDS sums the element matrices of the previous set into a global matrix.

Results are given for the Apollo, VAX, T800A, the TDS running sequential code, the TDS with a network of one T414 running parallel code, and the TDS with a network of 12 T414's running parallel code. The results show that, for 100 elements, a single T414 transputer runs faster than the Apollo, and a single T800 (running at only half the speed of production versions) is faster than the VAX. The step pattern in the 12 transputer curve is indicative of the work assignments in the parallel code. The network takes roughly the same amount of time to calculate 12 or fewer elements so the step slope is close to horizontal; a finite jump in execution time occurs after a multiple of twelve is exceeded, however, since after that point some transputers have more elements for which to calculate matrices than do others.

# EXECUTION TIMES FOR GAUSS-JORDAN MATRIX INVERSION

SPARTA, INC.



# Parallel Solution of Linear Equations

Gauss-Jordan elimination and Gauss-Jordan inversion, both with partial pivoting, were implemented on the transputer network. No attempts were made at taking advantage of the matrices' sparse, symmetric nature, so the implemented algorithms can be used to solve any well-conditioned set of linear equations.

Gauss-Jordan elimination and inversion can both be implemented on parallel processors with great efficiency since they can be easily load balanced, and since they are compute bound if the number of equations is much larger than the number of processors. The matrix is first divided up among the processors in the network. The matrix in the TBFES was subdivided by distributing entire matrix columns to the transputers, although several ohter schemes are also possible (row distribution, block distribution). Columns of the matrix must be distributed so that the work load is balanced. Elimination is slightly more complicated to load balance than inversion, since elimimation allows columns of the matrix to become inactive as the solution progresses from left to right. If some processors have only inactive columns, then they will become idle and lower the overall efficiency of the program. Columns in the TBFES were distributed in an alternating sequence so that at any stage during elimination, all processors will have equal numbers of active and inactive columns (plus or minus one column). For example, if 4 processors had to solve a system of 26 equations, the column distibution scheme would result in the following:

| Processor | Columns |
|---|---|
| 1 | 1, 5, 9, 13, 17, 21, 25 |
| 2 | 2, 6, 10, 14, 18, 22, 26, 27 |
| 3 | 3, 7, 11, 15, 19, 23 |
| 4 | 4, 8, 12, 16, 20, 24 |

The 27th column is the load vector and resides on the same node as the last coefficient column.

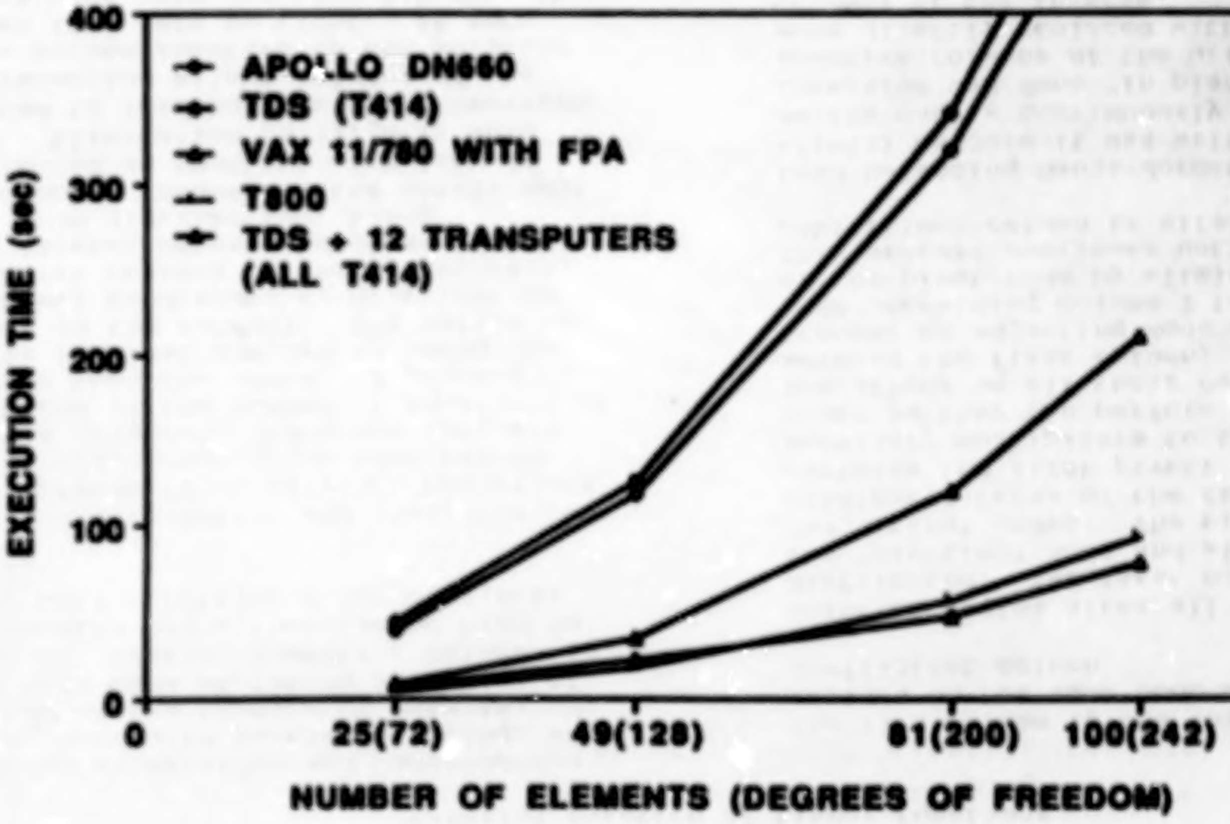
Solution begins after all columns are distributed. The first node then becomes the "pivoting" node and all others become "adjusting" nodes. The pivot node eliminates terms of the first column (which contains the first pivot) and sends the necessary multipliers to the adjusting nodes so they can perform the same operations on all their columns. After zeroing the first column, the pivot node becomes an adjusting node and tells the node containing column 2 to assume the role of the pivot node by eliminating column 2. This process continues until the last coefficient column is eliminated.

Load balancing Gauss-Jordan inversion was trivial because it was written to keep the entire matrix continuously active--inversion was done "in place" so that the inactive columns of the original matrix were directly replaced with the active columns of the inverse.

Efficiency of parallel equation solvers increases as the number of columns per processor increases, because the computational effort increases with the cube of the number of equations (= number of columns) while communication increases with the product of the number of processors and the number of equations. For extremely large systems of equations solved on a small network of parallel processors, communication is negligible compared to the enormous computational effort and efficiency should reach 100%.

# EXECUTION TIMES FOR COMPLETE FE PROBLEM

SPARTA, INC.



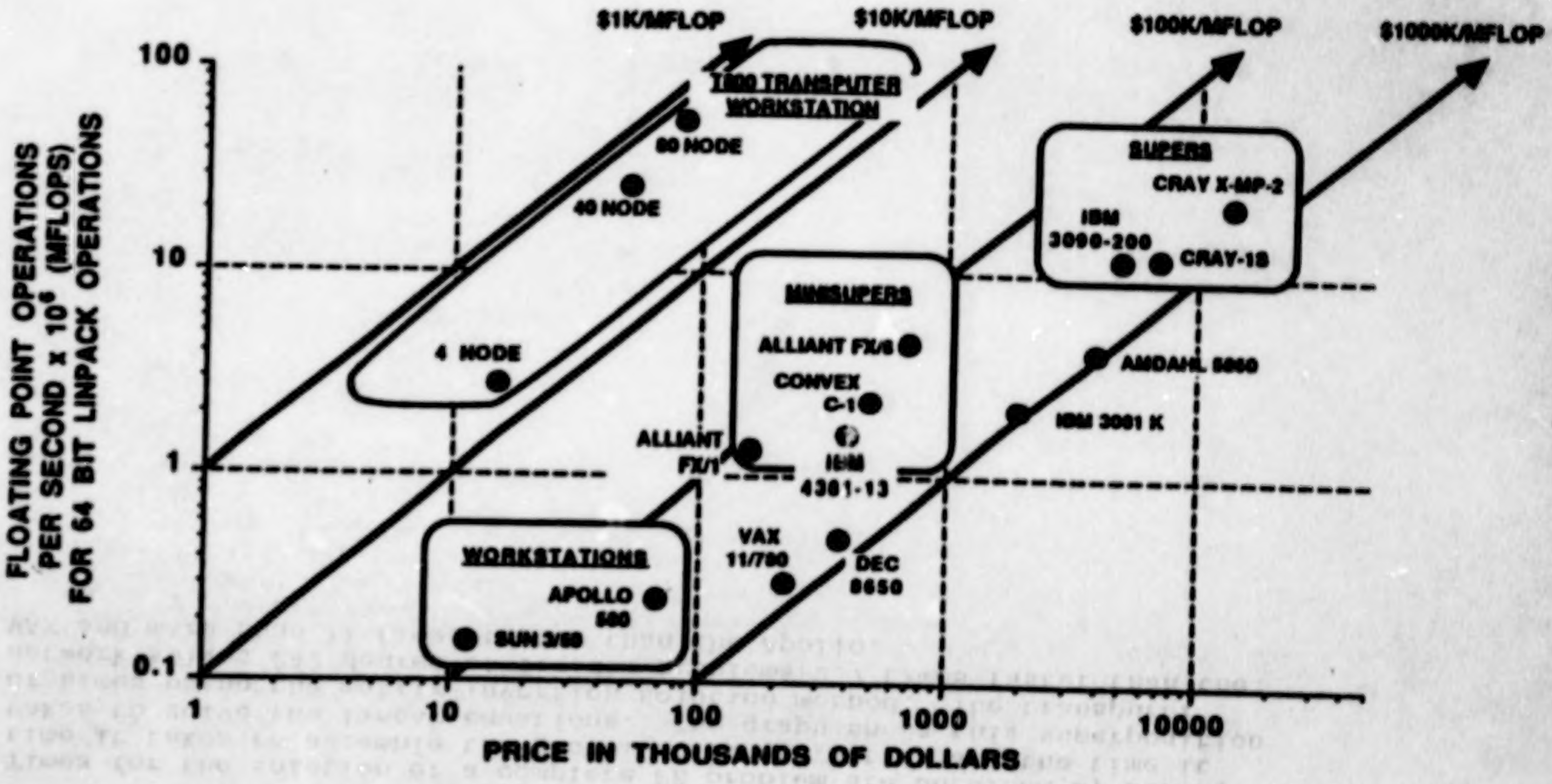
## Complete Finite Element Solution

Times for the solution of a complete FE problem are obtained by adding the time it takes to assemble the global stiffness matrix and the time it takes to solve the linear equations. The graph shows this superposition of times using the matrix inversion solution method. The transputer network solves 242 degree of freedom problems 2.7 times faster than the VAX and more than 11 times faster than the Apollo.

C-2

SPARTA, INC.

# PERFORMANCE VERSUS PRICE FOR SEVERAL COMPUTING SYSTEMS



ORIGINAL PAGE IS
OF POOR QUALITY

## A Large-Scale, Practical Transputer FE Solver

It is evident that transputers have enourmous potential; networks of T800's can be assembled at a rate of $1000 per Mflop, making supercomputer performance levels available at minicomputer prices. Finite element analysis is in a unique position to fully exploit the advantages offered by a large transputer network: the FE Method is compute-intensive and has inherent parallelism; and there is such a pressing need for more powerful finite element solving capabilities that an inexpensive, super-powerful FE solver would be in great demand in the scientific and engineering communities. This section addresses some issues critical to the implementation of an effective FE solver on a large network of transputers.

# VARIABLES AFFECTING PERFORMANCE OF PARALLEL FE SOLVERS

SPARTA, INC.

- PROCESSOR SPECIFICATIONS
  - MFLOPS AVAILABLE
  - NUMBER OF LINKS
  - COST OF PROCESSOR
  - COST ON IN-CORE MEMORY FOR EACH PROCESSOR
- INPUT/OUTPUT SPECIFICATIONS
  - LINK TRANSFER RATE
  - MASS STORAGE TRANSFER RATE
  - COST OF MASS STORAGE
- NETWORK GRANULARITY
- SOLUTION METHOD
  - DIRECT FACTORIZATION METHODS
  - FRONTAL METHOD
  - ITERATIVE METHODS

Many variables would influence the performance of a large scale parallel FE solver. These variables, which fall in four general categories (processor specifications, network granularity, input/output specifications, and the solution method used), must be balanced against each other to yield the optimum parallel solver. Finding the appropriate combination is difficult, especially since the slightest improvement in the implementation of a solution method may require a considerable hardware modification. The parameter balancing problem will most likely be iterative, but a feasible initial approach might be to allocate available funds so that a low granularity network (few processors, much RAM available to each processor) with enough memory to solve the largest problems is assembled. Two options are possible if the required amount of in-core memory is unobtainable for a low-granualarity network: external mass storage could be added to the processors (maintaining low granularity and using an out-of-core solver), or the number of processors could be increased until enough RAM is accumulated (allowing granularity to increase and using an in-core solver). The final hardware configuration is then used as a basis for selecting, modifying and/or designing a solution method.

SPARTA, INC.

# LARGE SCALE FE SOLVER REQUIRES MUCH MEMORY

## TWO APPROACHES TO SATISFYING MEMORY REQUIREMENTS

- **IN-CORE:**
  - ADD RAM TO EACH PROCESSOR
  - ADD PROCESSORS UNTIL ENOUGH RAM IS ACCUMULATED
- **OUT-OF-CORE: ADD MASS STORAGE DEVICES TO THE PROCESSOR NETWORK**

## Memory Requirements

Large-scale FE analysis requires extraordinary amounts of memory. Although transputer networks can be made so large that the collective in-core memory is sufficient to store and solve the entire problem (a $1,000,000 transputer network--1000 T800's with 1 Mbyte each--has 1 Gbyte of in-core memory, sufficient for a 100,000 degree of freedom double-precision problem stored with a half bandwidth of 1250), such high granularity networks require proportionately higher amounts of communication making high efficiency less obtainable. Mass storage devices added to relatively smaller networks of transputers create problems of their own. The slow throughput rates typical of external storage could starve the transputer network of data, making the network as slow as a sequential processor. The throughput rate could be increased by adding more external storage devices, but this quicky raises the cost of a parallel FE solver. If throughput rates have to be so high that each processor requires its own storage device (as might be necessary for an iterative solution method), the price of such a network would keep its size, and therefore its power, low.

# EXISTING SOLUTION METHODS HAVE DRAWBACKS

SPARTA, INC.

- DIRECT METHODS
  - GAUSS-JORDAN ELIMINATION
    - -- EFFICIENT PARALLEL IMPLEMENTATION
    - -- INEFFICIENT FOR FE ANALYSIS

  - LUD DECOMPOSITION
    - -- BAND METHOD HAS EFFICIENT PARALLEL IMPLEMENTATION IF BW >> # Xp's
      - DOES NOT TAKE ADVANTAGE OF SPARSENESS WITHIN THE BAND
      - DOES NOT TAKE ADVANTAGE OF LOCAL VARIATION IN BANDWIDTH
    - -- ENVELOPE AND BLOCK METHODS ARE MORE EFFICIENT, BUT
      - INCREASE IN EFFICIENCY IS PROBLEM DEPENDENT
      - LOAD BALANCING BECOMES PROBLEM DEPENDENT

  - FRONTAL METHOD
    - -- LOW OPERATIONS COUNT
    - -- VARIABLE FRONTWIDTH MAKES LOAD BALANCING DIFFICULT
    - -- ADDITIONAL BOOKKEEPING MAY BE PROHIBITIVE

## Selecting a Solution Method--Load balancing, Communications and Efficiency

Tapping the full potential of a large transputer network is possible only if (1) each processor contributes to the solution of the entire problem and is never idle, and (2) the processors spend nearly all their time performing useful calculations rather than transmitting information to other processors. These two criteria which determine the overall efficiency of parallel solvers are often difficult to satisfy simultaneously. Although one part of the FE Method, the generation of element stiffness matrices, is so parallel that it can easily satisfy the conditions governing efficiency, the more time-consuming part of solving the structural equations does not readily yield to efficient implementation. A few of the advantages and disadvantages of several solution methods to structural equations (having sparse, symmetric, banded, positive-definite coefficient matrices) are given below:

- Gaussian Elimination: High efficiency only if the bandwidth is much larger than the number of processors. A new set of constraints requires the entire problem to be solved again.
- Gauss-Jordan Elimination: Easily load balanced, but has a high operations count. A new set of constraints requires the entire problem to be solved again.
- LU or Cholesky Decomposition: Factorization methods are preferable to straightforward elimination since new constraints can be solved for in a quick back-substitution step. A few of its variations are described below:
  - Band Method: Efficient only if the bandwidth is much larger than the number of processors. Since the band is always of a known geometry, it is easy to divide up among processors for load balancing. This method does not take advantage of sparseness within the band, or of local bandwidth variation, so it is not the best of LU methods.
  - Envelope, Block & Skyline Methods: These methods usually have a lower operations count than the Band Method since they take advantage of variations in the band, but for the same reason become difficult to load balance. The matrix for these methods is highly irregular and static load balancing schemes are not obvious. Dynamic load balancing might be used, but this would add to the communications load.
- Frontal Method: This method typically has a very low operations count but is difficult to load balance (the front width continuously changes) and requires complicated bookkeeping for parallel implementation. The frontal method on a sequential machine has high bookkeeping overhead to begin with, so the additional expense of parallel logic overhead may be prohibitive.

SPARTA, INC.

# EXISTING SOLUTION METHODS HAVE DRAWBACKS

- ITERATIVE METHODS
  - EFFECTIVE FOR HIGH ACCURACY, 3-D ANALYSIS, ADAPTIVE ANALYSIS
  - REQUIRE MUCH COMMUNICATION
  - LARGE PROBLEMS REQUIRE FREQUENT ACCESS TO MASS STORAGE

**Iterative Methods (Conjugate Gradient Method, Incomplete Cholesky Decomposition, SOR Method):** Iterative methods are effective for three dimensional and higher degree element problems, problems requiring high accuracy, and adaptive mesh refinement solution methods, but typically require more interprocessor communication than direct methods. Problems too large to fit in the collective core memory of the processor network demand high throughput rates to external storage since the entire matrix must be updated at every iteration. Data transfer rates to mass storage devices could easily become the design criteria for large scale iterative solvers.

# FIVE TRANSPUTER FE SOLVERS COSTING $200,000

SPARTA, INC.

| | ALL 256Kb T800's | ALL 1Mb T800's | 100 1M T800's + 9 DISKS | 67 1M T800'S + 19 DISKS | 30 1M T800's + 30 DISKS |
|---|---|---|---|---|---|
| NUMBER OF PROCESSORS | 200 | 133 | 100 | 67 | 30 |
| MFLOPS | 300 | 200 | 150 | 100 | 45 |
| NETWORK RAM | 50 Mb | 133 Mb | 100 Mb | 67 Mb | 30 Mb |
| EXTERNAL MEMORY | --- | --- | 1,620 Mb | 3,420 Mb | 5,400 Mb |
| LARGEST PROBLEM (D.O.F.) | 8,095 | 13,200 | 47,400 | 67,600 | 84,300 |
| TIME TO SOLVE 8,000 D.O.F. | 36 sec | 53 sec | 71 sec | 106 sec | 237 sec |
| TIME TO SOLVE LARGEST PROBLEM | 37 sec | 240 sec | 4.1 hrs | 17.8 hrs | 77 hrs |

## Estimates of Performance for Five Transputer FE Solvers Costing $200,000

In order to estimate the performance levels that could be obtained from a TBFES, an analysis was made of five transputer hardware configurations, each costing $200,000. The five configurations varied by (1) the amount of RAM available to each transputer and (2) the amount of external storage available to the network. Banded LUD decomposition was used as the common solution method since its execution times can be predicted with much greater accuracy that frontal or iterative methods.

The following assumptions were used to predict execution times for the five transputer FE solvers:

Notation:
N = degrees of freedom
Xp = number of transputers
matrix operation = one addition and one multiplication

Prices:
T800 with 256 Kbytes = $1,000
T800 with 1 Mbyte = $1,500
180 Mbyte SCSI disk drive with transputer interface = $5,200

Assumptions:
Solution method = Banded LU decomposition
Half bandwidth = 0.1*N
double precision storage requirement in bytes = 8 * 0.1N * N = 0.8N*N
T800 performance for double precision matrix operations, including overhead = 0.6 Mflop
number of matrix operations for banded LU decomposition = 0.1N * (N*N)/12 = (N*N*N)/120
time for LU decomposition = (N*N*N)/(120 * 600,000 * Xp)
efficiency = 100%
I/O transfer rates to disk do not impede performance

The estimates are based on rough calculations, but can still be used to qualitatively compare the different configurations. The solver with the most transputers, and therefore the most Mflops, appears to be the fastest of the five FE solvers, but its high granularity suggests that the 100% efficiency assumption probably will not apply. The 200 transputer network also has relatively little memory so it can only be used for problems with fewer than 6,000 degrees of freedom. The FE solver at the other end of the spectrum, the 30 transputers with a disk drive apiece, has such low granularity and such a high transfer rate to external storage that high efficiency is certain. Such a configuration would be useful for solving the largest problems but is four times slower than the 133 transputer system which can also solve problems of considerable size. The optimum combination depends entirely on user requirements; if only small problems are anticipated, or if solution speed is critical, the 133 transputer network seems ideal. A demand for the solution of very large problems would most likely call for the network with the most external storage.

SPARTA, INC.

# CONCLUSION

- SBIR STUDY DEMONSTRATED FEASIBILITY OF A TFES
- PROJECTION OF RESULTS SUGGESTS A LARGE-SCALE TFES WITH SUPERCOMPUTER POWER COULD BE BUILT FOR THE PRICE OF A MINICOMPUTER
- TAPPING ALL THE POTENTIAL POWER OF A TRANSPUTER NETWORK IN A LARGE-SCALE TFES
  - APPEARS ACHIEVABLE
  - REQUIRES INTENSIVE RESEARCH AND DEVELOPMENT EFFORT
  - COULD REVOLUTIONIZE FE ANALYSIS

## Conclusion

The SBIR study determined that the implementation of the FE Method on a network of transputers is highly feasible. Reasonable projections suggest that a large scale FE solver with Cray-level power could be assembled for $100,000. An intensive research and development effort is required to effectively implement such a powerful FE solver, but the result could revolutionize finite element analysis by making supercomputer analysis capacities widely available to scientific, research and engineering communities.

N89
29778

UNCLAS

ND315753

N89-29778

S5-39
288
311946

# TRANSPUTER PARALLEL PROCESSING AT NASA LEWIS RESEARCH CENTER

By

Graham K. Ellis
Institute for Computational Mechanics in Propulsion
NASA Lewis Research Center
Cleveland, Ohio

## ABSTRACT

The transputer parallel processing lab at NASA Lewis Research Center (LeRC) consists of 69 processors (transputers) that can be connected into various networks for use in general purpose concurrent processing applications. The main goal of the lab is to develop concurrent scientific and engineering application programs that will take advantage of the computational speed increases available on a parallel processor over the traditional sequential processor.

Current research involves the development of basic programming tools. These tools will help standardize program interfaces to specific hardware by providing a set of common libraries for applications programmers.

The thrust of the current effort is in developing a set of tools for graphics rendering/animation. The applications programmer currently has two options for on-screen plotting. One option can be used for static graphics displays and the other can be used for animated motion.

The option for static display involves the use of 2-D graphics primitives that can be called from within an application program. These routines perform the standard 2-D geometric graphics operations in real-coordinate space as well as allowing multiple windows on a single screen. These real-coordinate routines can be used stand-alone for static displays or with the graphics engine, which is discussed below, for animated graphics.

For animation, a high-performance graphics engine has been developed. The graphics engine consists of 18 transputers connected in a 2-D mesh arrangement. Each node in the network performs a single graphics primitive computation. Frequently requested tasks such as line clipping can be performed on multiple nodes. The distribution of the normal display processor workload onto the distributed network increases the throughput by spreading the computationally intensive tasks over many processors.

The use of the graphics engine is transparent to the applications programmer other than requiring the inclusion of the appropriate pre-compiled code in the application program.

Future research targeted for the transputer lab includes parallelization of optimal and state-variable feedback control systems for simulating the control of both steady-state and transient vibration in rotor-dynamic systems.

**PRECEDING PAGE BLANK NOT FILMED**

**PAGE 106 INTENTIONALLY BLANK**

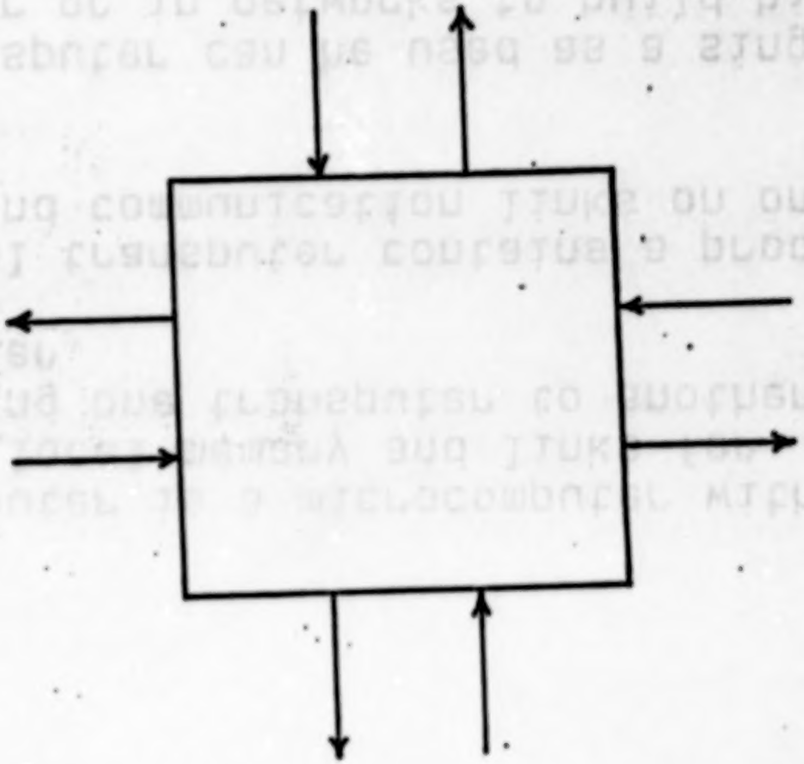
# OVERVIEW

- Transputer hardware and software
- Graphics software development
- Future activities

A transputer is a microcomputer with its own local memory and links for connecting one transputer to another transputer.

A typical transputer contains a processor, memory and communication links on one chip.

The transputer can be used as a single chip processor or in networks to build high performance concurrent systems.



The transputer link arrangement.

The software building block is the process.

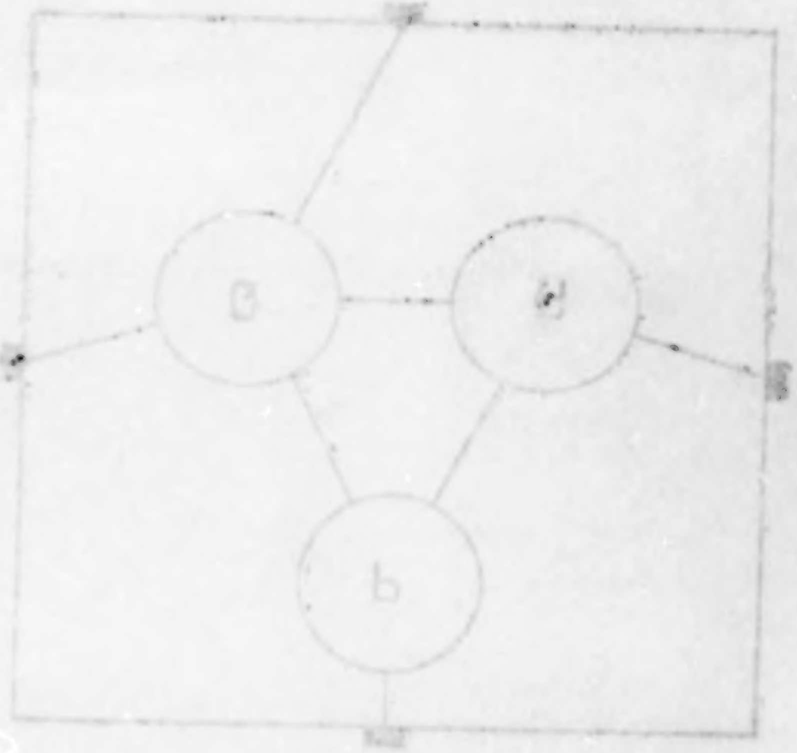
Each system is designed in terms of an interconnected set of processes.

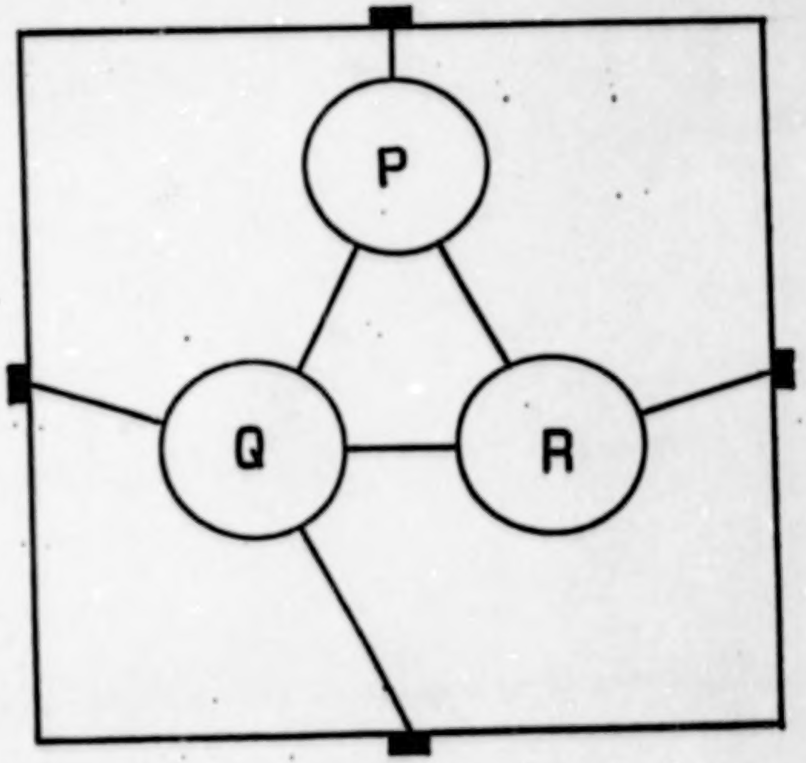
Each process can be regarded as an independent unit of design. It communicates with other processes along point-to-point channels.

PRECEDING PAGE BLANK NOT FILMED

PAGE 111 INTENTIONALLY BLANK

The transputer links are arranged as four pairs of input and output channels. These channel pairs allow the connection of large multi-transputer networks for parallel solution of computationally intensive problems.

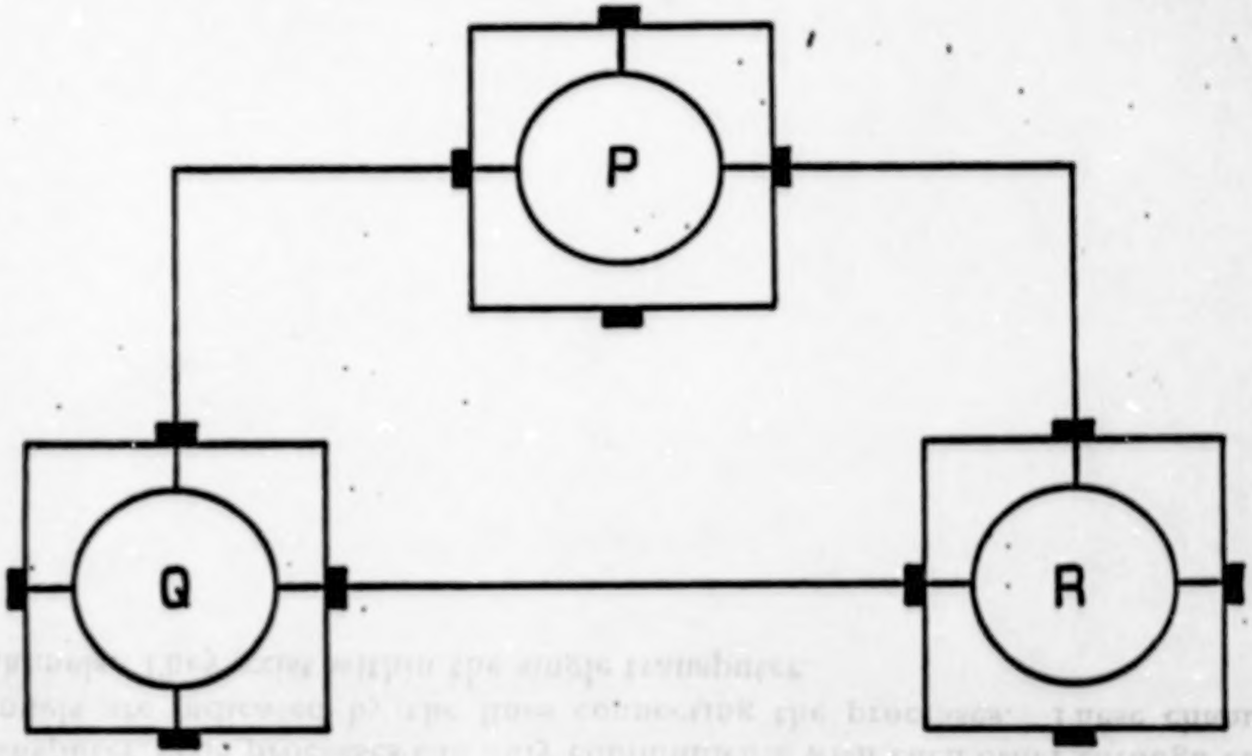




A PROGRAM ON A SINGLE TRANSPUTER

# A PROGRAM ON A SINGLE TRANSPUTER

The processes P, Q, and R are shown running in parallel (simulated with time-slicing) on a single transputer. The processes can only communicate with each other through channels. These channels are indicated by the lines connecting the processes. These channels are internal channels. They exist within the single transputer.



THE SAME PROGRAM ON THREE TRANSPUTERS

## THE SAME PROGRAM ON THREE TRANSPUTERS

**The processes P, Q, and R have now been distributed on a network of three transputers. These channels are physically placed on the links that connect the transputers in the network. Notice the lines representing the channels (and links in this case) connect the network in the same configuration shown above.**

# TRANSPUTER HARDWARE

- 40 T414 32-bit integer processors, 256KBytes per node (upgrade to T800 floating point chip 1/88)
- 27 T212 16-bit integer processors 24 with 8KBytes high-speed memory, 3 with 64KBytes
- 1 T414 based medium performance graphics board with 512x512 pixel resolution, 256 colors out of 256,000
- 1 T414 based development board, plugs into IBM PC slot, 2MBytes memory. System development software runs here

## TRANSPUTER HARDWARE

The transputer lab hardware is as described here. The development system, a transputer-based card that plugs into an IBM compatible PC/XT or AT, is where the software development is performed. The other facilities are used only when performing an analysis or simulation.

## OBJECTIVE

- Animate structural model vibration response

# APPROACH

- 2-D graphics primitive library for applications programs
- Distribute display processor workload on a network of transputers for increased throughput (graphics engine)

121 INTENTIONALLY BLANK

INTENTIONALLY BLANK

PRECEDING PAGE BLANK NOT FILMED

## APPROACH

In order to generate plots, a library of 2-D routines for application programs has been developed. For animation, the 2-D routines above can be used with a graphics engine. This graphics engine consists of a network of 18 transputers on which the normal computational load of the display buffer is distributed.

# 2-D APPLICATION PRIMITIVES

- Graphics transformations
- Screen and window manipulation
- Relative coordinate commands
- Absolute coordinate commands

## 2-D APPLICATION PRIMITIVES

**The following 2-D applications programs have been developed:**

**Graphics transformations**

- **scale**
- **rotate**
- **translate**
- **make.identity**
- **combine.transformations**
- **transform.points**
- **map.to.screen.coords**

**Screen and window manipulation**

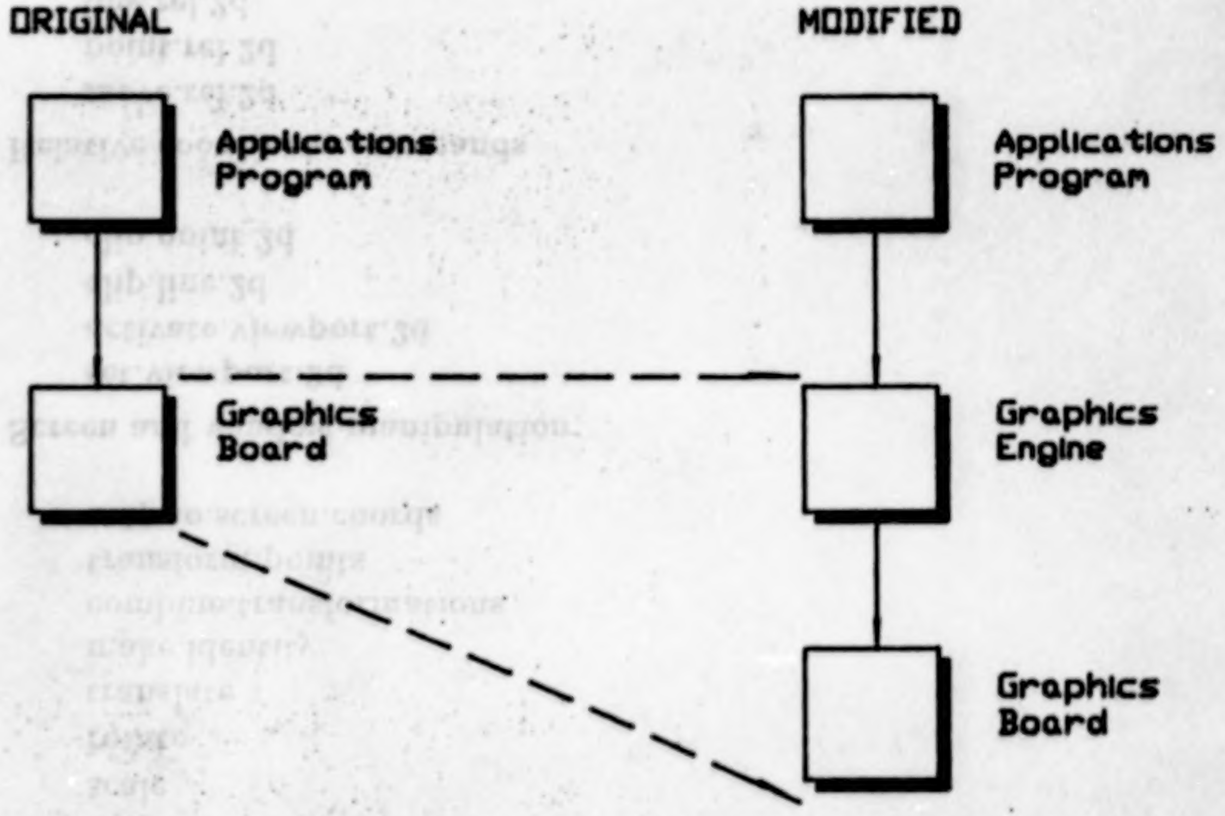
- **set.viewport.2d**
- **activate.viewport.2d**
- **clip.line.2d**
- **clip.point.2d**

**Relative coordinate commands**

- **move.rel.2d**
- **point.rel.2d**
- **line.rel.2d**

**Absolute coordinate commands**

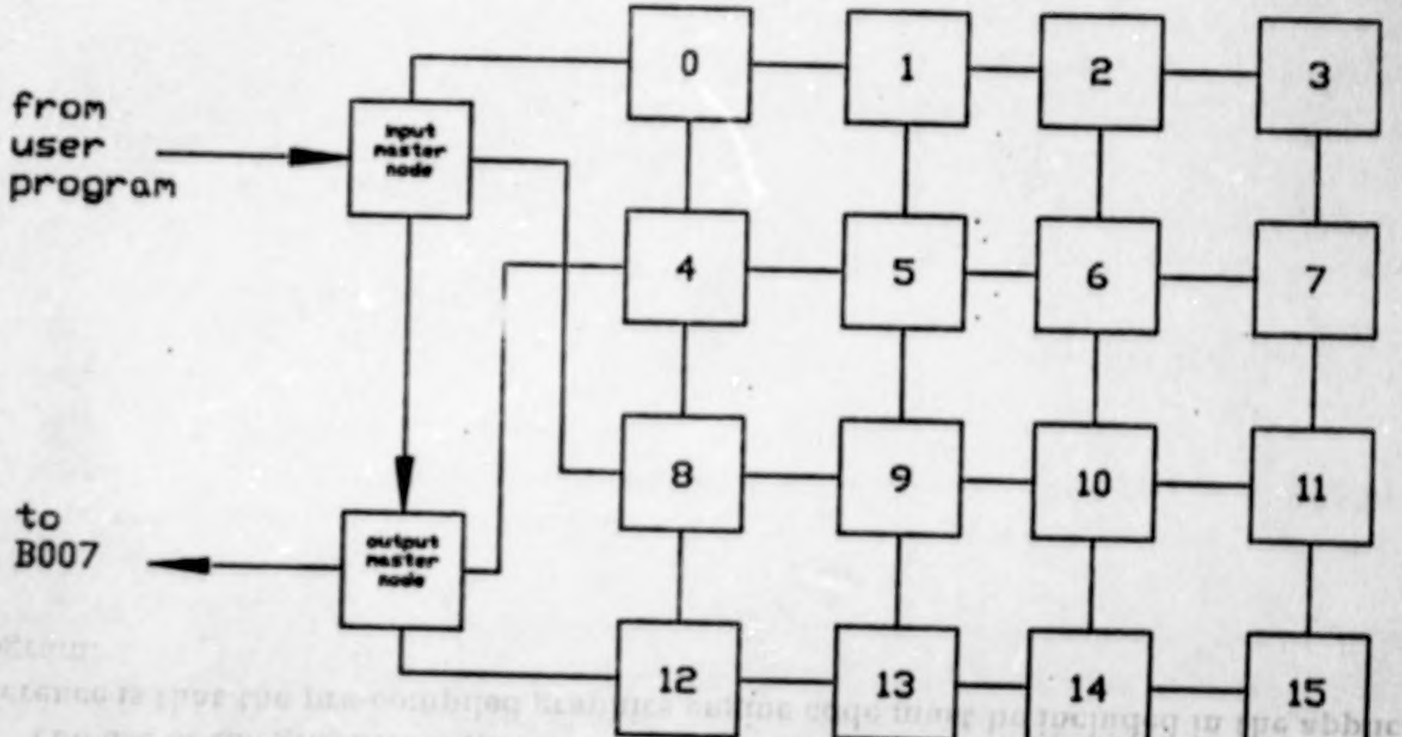
- **move.abs.2d**
- **point.abs.2d**
- **line.abs.2d**


ORIGINAL
Applications Program
Graphics Board
MODIFIED
Applications Program
Graphics Engine
Graphics Board

Graphics Engine Implementation. Transparent to Applications Programs.

## GRAPHICS ENGINE IMPLEMENTATION. TRANSPARENT TO APPLICATIONS PROGRAMS.

The use of the graphics engine is transparent to the applications programmer. The only difference is that the pre-compiled graphics engine code must be included in the applications program.


from
user
program
input master node
output master node
to
B007
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Multiple Processor Graphics Display Engine

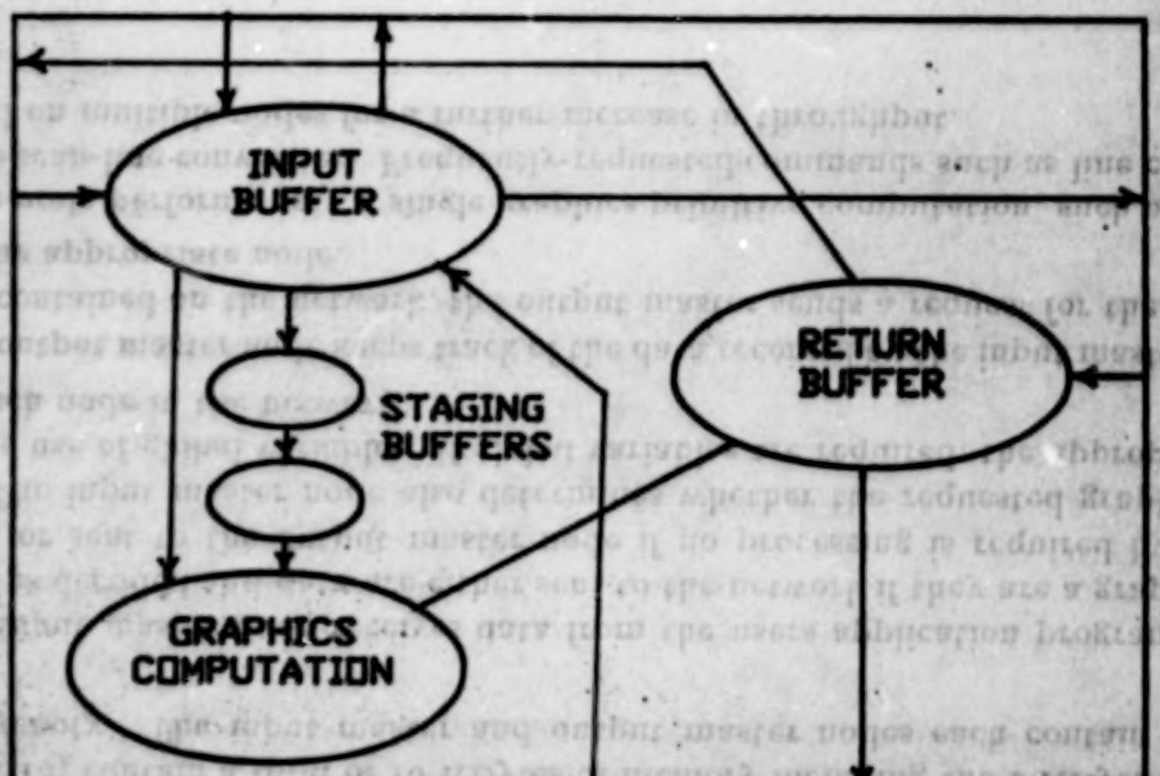
# MULTIPLE PROCESSOR GRAPHICS DISPLAY ENGINE

The graphics engine consists of 18 20-MHz T212 16-bit integer transputers connected in a 2-D mesh arrangement. The 16 processors that make up the computational nodes (nodes 0 through 15) contain a total of 10 KBytes of memory including the 2 KBytes of transputer onchip memory. The input master and output master nodes each contain 64 KBytes of memory.

The input master node receives data from the users application program. A graphics command is decoded and data are either sent to the network if they are a graphics primitive operation or sent to the output master node if no processing is required by the graphics engine. The input master node also determines whether the requested graphics command entails the use of global variables. If global variables are required, the appropriate data are sent to each node in the network.

The output master node keeps track of the data received by the input master node. If the data are contained on the network, the output master sends a request for the pre-processed data to the appropriate node.

Each node performs only a single graphics primitive computation, such as line clipping and circle scan-line conversion. Frequently-requested commands such as line clipping can be performed on multiple nodes for a further increase in throughput.


INPUT
BUFFER
STAGING
BUFFERS
RETURN
BUFFER
GRAPHICS
COMPUTATION

GRAPHICS ENGINE NODE BUFFERS

## GRAPHICS ENGINE NODE BUFFERS

Each node in the graphics engine contains several concurrently executing processes. The routing of data through the network is performed by the input and return buffers. The staging buffers store pending requests for the graphics computation process. The graphics primitive computation process performs its computation and waits for a signal to send the data back to the output master node through the return buffer.

Data routing decisions are made both in the input and return buffers.

# KEY PROBLEMS

- How to handle global variables on a distributed network
- How to maintain computational synchronization on a distributed network
- How to avoid communication deadlock on the network

# SOLUTIONS

- Each computational node has input and return buffers to intelligently communicate on the network
- All data sent on the network is tagged so the buffers can make the appropriate routing decisions
- Each node only performs a single graphics primitive computation. Frequently requested commands can reside on multiple nodes

# SOLUTIONS

- Synchronization is maintained by the output master control node. It requests results from the network in the same order as the user application sent its requests for work
- Deadlock is avoided by using 'staging buffers' on each node to store pending work requests. The input buffer controls the number of work requests sent to each node so the number of pending work requests is never larger than the number of staging buffers

# FUTURE ACTIVITIES

- Parallel algorithms for optimal and state-variable feedback controls applications.
- Applications include control of unbalance forces in rotating machinery and control of transient vibrations

# REFERENCES

1. Transputer Reference Manual, INMOS, Ltd., October 27, 1986.

N89
29779
UNCLAS

# INNOVATIVE ARCHITECTURES FOR DENSE MULTI-MICROPROCESSOR COMPUTERS

Dr. Robert E. Larson
Chairman and C. E. O.
Expert-EASE Systems, Inc.
Belmont, California 94002

The purpose ~~of this presentation~~ is to summarize a Phase I SBIR project performed for the NASA/Langley Computational Structural Mechanics Group. The project was performed from February to August 1987.

The main objectives of the project were to:

1. Expand upon previous research into the application of "chordal ring" architectures to the general problem of designing multi-microcomputer architectures.
2. Attempt to identify a family of chordal rings such that each chordal ring can be simply expanded to produce the next member of the family.
3. Perform a preliminary, high-level design of an expandable multi-microprocessor computer based upon chordal rings.
4. Analyze the potential use of chordal ring based multi-microprocessors for sparse matrix problems and other applications arising in computational structural mechanics.

N89-29779

56-39
348.
211947
E922 7830

# PRESENTATION OVERVIEW

- Corporate Capabilities
- Transputer Hardware
- Interconnection Architecture
- Applications
- User Interface

## PRESENTATION OVERVIEW

This presentation covers the following topics:

1. The Corporate Capabilities of Expert Ease Systems. Particular emphasis will be placed on the company's driving objective to use projects such as this one as stepping stones to develop products which can compete and thrive in the American and International marketplaces.

2. Transputer Hardware Specifications. The T800 Transputer microprocessor recently introduced by Inmos Corporation is a very powerful chip that is ideally suited for multi-microprocessor architectures. Inmos has published their Transputer specifications widely, and a substantial amount of information can be obtained directly from the company.

3. Interconnection Architecture. The critical goal of this Phase I SBIR project was to investigate novel interconnection architectures called "chordal rings". Chordal rings have several very interesting properties that make them particularly competitive with other architectures commonly cited as potential multi-microprocessor architectures, especially the hypercube upon which Intel has based its PSC. In particular, chordal ring "families" have been identified. Each chordal ring in the family can be easily expanded into the next larger ring in the family.

4. Applications of a Choral Ring-Based Multi-Microprocessor. The talk presents a summary of several interesting examples of how a chordal ring computer could be used to solve problems of interest in computational structural mechanics.

5. User Interface. Expert-EASE Systems has several software products which incorporate powerful, user-friendly interfaces. Such interfaces are emphasized in our projects for both corporate and government clients. Indeed, the development of this type of interface has been emphasized in our Phase II proposal to develop a chordal ring-based multi-microprocessor which can be expanded from 10 to over 1000 T800 Transputer chips.

In summary, then, I will summarize the substantial progress made during Phase I, especially how we have demonstrated the feasibility and advantages of a multi-microprocessor machine based upon chordal ring architecture.

# CORPORATE OVERVIEW

## Expert-EASE Systems

- o Product-Oriented Company
- o Expertise
  - Distributed Computing
  - Software Design
  - User Interfaces
  - Artificial Intelligence
- o Previous Phase II Successes
  - Pantheon
  - EASE+NEXPERT

## Phase II Project Managers

- o Dr. Robert E. Larson, Principal Investigator
  - 1982 IEEE President
  - Founder/President of Systems Control, Inc., Palo Alto, California
  - Stanford Professor
- o Dr. John G. O'Reilly, Assistant Principal Investigator
  - DMS for Space Station
  - Extensive Networking and Real-Time Computing Experience
- o Dr. Sidney Fernbach, Applications Consultant
  - Former Head, Physics and Computing, LLL
  - Supercomputer Expert (DOD, DOE, NSF)

# CORPORATE OVERVIEW

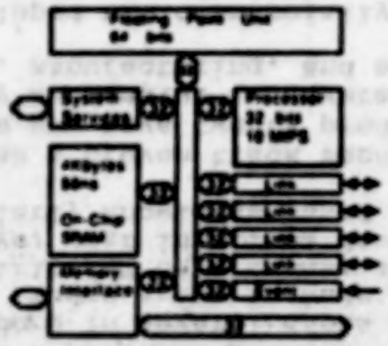
Expert-EASE Systems is a product-oriented company with strong personnel and a proven track record in several areas of particular interest to researchers in computational structural mechanics. These areas include: distributed computing, especially the design of distributed architectures and distributed data bases; software design for all types of computers, from microcomputers to Crays; user interface design, including natural language interfaces; and artificial intelligence, particularly knowledge-based expert systems.

Expert-EASE Systems has a proven track record of developing and marketing commercially available software products. We now have twenty products in our "EASE+" family of microcomputer based programs. These have proven very successful in several industries, including electric utilities, process control, defense and aerospace, manufacturing, and artificial intelligence.

Further, we have developed two commercially successful products as a direct consequence of the SBIR program. First, we developed a natural language program to simultaneously access multiple data bases residing on multiple heterogeneous computers. This product, called Pantheon, led directly to the formation of a subsidiary company, Pantheon Systems. Second, a DOE Phase II award led to the integration of our EASE+ interface with a widely used expert system, NEXPERT, produced by Neuron Data, Inc. We estimate that this product, which we call EASE+NEXPERT, will have first year sales in excess of $1,000,000.

Now that I have presented an introduction to our general capabilities, I'd like to present our specific qualifications to perform our proposed Phase II effort. The effort will involve three managers, each of whom has years of experience developing computers based upon novel architectures. The managers include: myself, I have spent over 25 years working on both distributed and centralized approaches to solving computationally intensive problems in many different fields; Dr. John G. O'Reilly, a distributed computer expert with whom I've worked for nearly 10 years and have had the opportunity to coauthor two books on distributed computing; and Dr. Sidney Fernbach who is generally considered one of the world's leading experts on supercomputers.

# IMS T800 FLOATING POINT TRANSPUTER


Floating Point Unit
64 bits
System Services
Processor
32 bits
10 MIPS
4K Bytes
On-Chip SRAM
Memory Interface
Link
Link
Link
Link
Event

- 32 - bit processor
- 15 MIPS
- 64-bit on-chip floating point processor
- 4K on-chip high speed RAM
- Four 20Mbits/sec communication links. ( On-chip )
- Sustained 2.25 MFLOPS (30 Mhz).
- On-chip memory controller

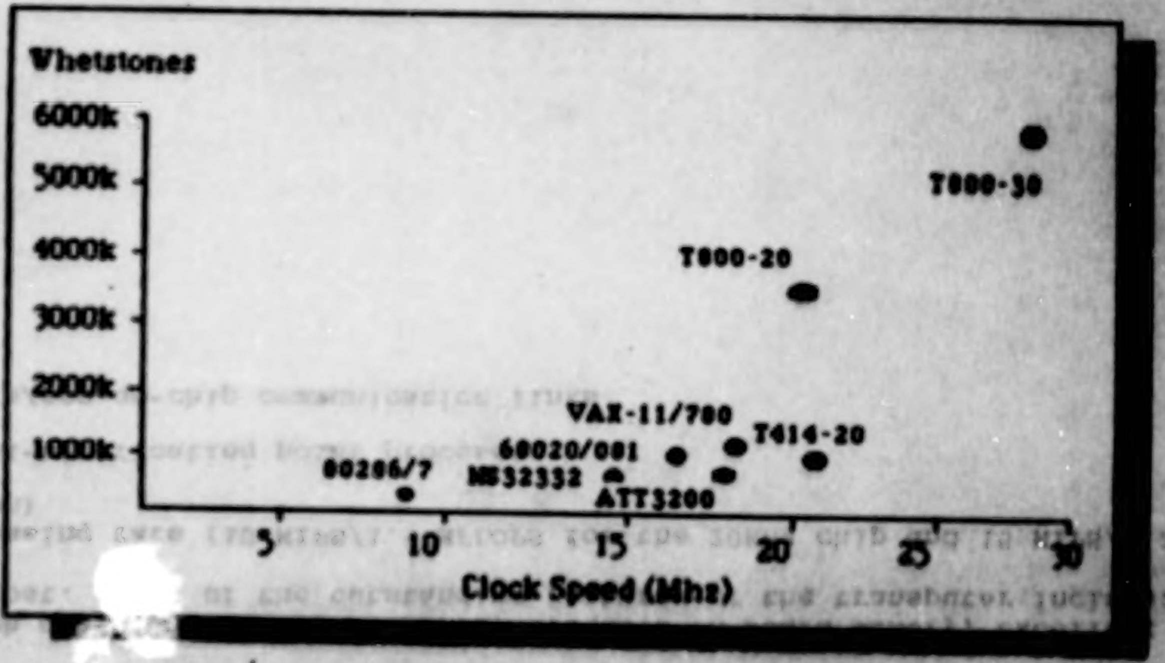
## IMS T800 FLOATING POINT TRANSPUTER

After extensive evaluations of a number of leading multiprocessor chips, the Inmos T800 transputer was selected as the basis of the choral ring-based multi-microcomputer to be built during Phase II.

The T800 combines high computational performance, adequate on-board memory, excellent communications facilities, and low cost. Some of the outstanding features of the transputer include:

- A high processing rate (10 MIPS/1.5 MFLOPS for the 20MHz chip and 15 MIPS/2.25 MFLOPS for the 30MHz version)
- An on-chip 64-bit floating point processor
- Four 20 Mbits/sec on-chip communication links

# T800 FLOATING POINT PERFORMANCE



## T800 FLOATING POINT PERFORMANCE

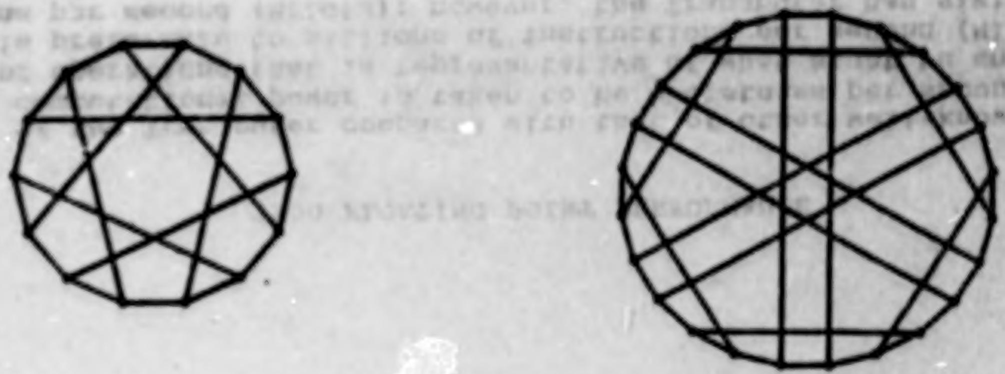
The computational power of the Transputer compares with that of other well-known microprocessors in this figure. The measure of computational power is taken to be Whetstones per second, which considers the execution of a mixture of operations that is representative of what might be encountered in an actual program. This measure is preferable to millions of instructions per second (MIPS) or millions of floating point operations per second (MFLOPS); however, the Transputer has similar advantage in terms of these measures.

As can be seen from the figure, the 30 MHz T800 (denoted as T800-30) and the 20 MHz T800 (denoted as T800-20) are both far more powerful than the leading microprocessors made by Intel (80286/87), Motorola (68020/081), National Semiconductor (NS32332) and AT&T (ATT 3200). Note that these microprocessors are also far more powerful than the Digital Equipment Corporation VAX 11/780, a well-known minicomputer. Finally, the T800-20 and T800-30 are substantially superior to the previous generation Transputer, the T414-20.

# PROPERTIES OF CHORDAL RINGS

n= Nodes - Number of microprocessors in network
k= Diameter - Maximum distance between any two nodes.
d= Degree - Number of communication channels on each node.

- Shorter network diameter with larger number of nodes.
- Even distribution of network traffic.
- Efficient use of resources.



## PROPERTIES OF THE CHORDAL RINGS

The investigation of multi-microprocessor architectures is typically performed as a graph theoretic problem in which the objective is to find a way in which n nodes can be interconnected so as to maximize some quantitative function of the architecture. In this formulation, each node represents a single processor and each link represents a communications channel connecting two processors.

Three terms from graph theory are commonly used to describe computer architectures:

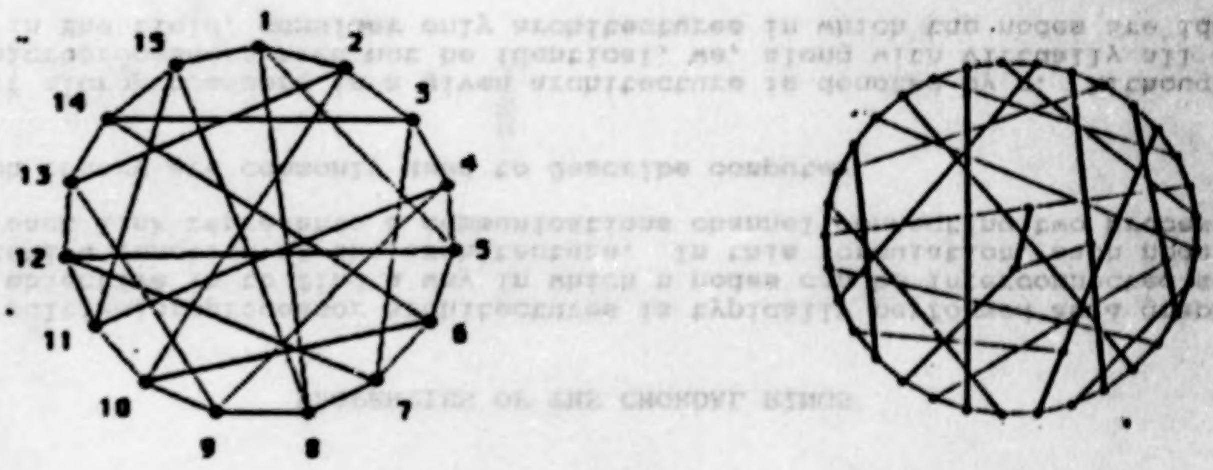
- The number of microprocessors in a given architecture is denoted by n. Although the individual microprocessors need not be identical, we, along with virtually all other researchers in the field, consider only architectures in which the nodes are identical.
- The diameter, k, of a graph is the number of links separating the two nodes which are furthest apart in the graph.
- The degree, d, of a graph is the number of nodes to which each node is connected. Again, in our research, we have assumed that all nodes in the graph are connected to the same number of other nodes in the graph.

Chordal rings have several properties which are of high value from the perspective of a multi-microprocessor architecture. Specifically, chordal rings:

- Provide large n for fixed values of d and k
- Allow for the even distribution of network traffic
- Provide for the well-balanced, efficient use of network resources, including node throughput and link bandwidth

# MODIFIED CHORDAL RING

- Maintains significant characteristics of Chordal Ring
- Easily expandable with minimal reconfiguration
- Uniform growth path (10,15,36,78...).


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

## MODIFIED CHORDAL RING

Our research was not limited to pure chordal rings, which, by definition possess rotational symmetry. Instead, we emphasized the analysis of "modified chordal rings" which: (a) (left figure) may or may not possess rotational symmetry, and (b) may not even be rings (right figure).

It is by breaking away from the constraints imposed by pure chordal rings that we were able to derive graphs with particularly good architectural attributes. Indeed, it was found that:

- Modified chordal rings provide virtually all of the good attributes of pure chordal rings, while providing substantially higher values of n for fixed values of d and k.
- Modified chordal rings can be assembled into families in which a given graph can be expanded to the next larger graph in the family by the addition of nodes and links in a reasonably well-defined manner.
- The growth path for these modified chordal rings can be selected to have a high level of uniformity. From a commercial marketing aspect, this permits computers to be designed so that they can be continually expanded by the user in appropriate increments when his needs require.

# COMPARISON OF CHORDAL RINGS AND HYPERCUBES

## - REDUCED DATA COMMUNICATIONS -

| | | **Hypercube** | | **Chordal Ring** | |
|---|---|---|---|---|---|
| **n** | **d** | **k** | **<p>** | **k** | **<p>** |
| 8 | 3 | 3 | 1.5 | 2 | 1.375 |
| 16 | 4 | 4 | 2.0 | 3 | 1.719 |
| 32 | 5 | 5 | 2.5 | 3 | 2.016 |
| 64 | 6 | 6 | 3.0 | 3 | 2.328 |
| 128 | 7 | 7 | 3.5 | 4 | 2.500 |
| 256 | 8 | 8 | 4.0 | 4 | 2.776 |

**PARAMETERS CONSIDERED**

## COMPARISON OF CHORDAL RINGS AND HYPERCUBES

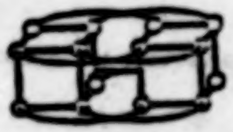
This chart is fairly self-explanatory. However, the main conclusion which should be drawn is:

- For fixed n and d, the chordal ring has significantly lower values of k and d (mean distance between nodes) than the corresponding hypercube.

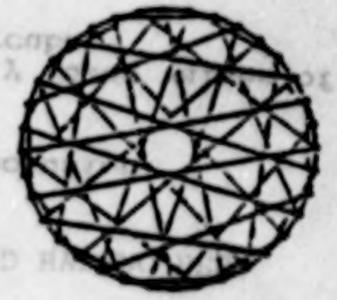
# COMPARISON OF CHORDAL RINGS AND HYPERCUBES

## - INCREASED PROCESSING POWER (2.25 X) -

| Level | Hypercube |  | New Chordal Ring Family |  |
|---|---|---|---|---|
|  | Nodes | Diameter | Nodes | Diameter |
| 1 | 8 | 3 | 10 | 2 |
| 2 | 16 | 4 | 15 | 2 |
| 3 | 32 | 5 | 36 | 3 |
| 4 | 64 | 6 | 78 | 4 |
| 5 | 128 | 7 | 166 | 5 |
| 6 | 256 | 8 | 348 | 6 |
| 7 | 512 | 9 | 742 | 7 |



d=4, k=4 HYPERCUBE



d=4, k=3 CHORDAL RING

## COMPARISON OF CHORDAL RINGS AND HYPERCUBES

This chart compares the new family of chordal rings with the standard hypercube graphs. It is clearly evident that for approximately equal computing power (i.e., numbers of nodes) the chordal rings possess substantially reduced diameter, and hence, would have fewer messages relayed between nodes.

The chart could also be interpreted with respect to a fixed diameter. That is, given a hypercube and a chordal ring with the same diameter, the chordal ring possesses substantially increased computing power via its many more nodes.

# APPLICATIONS

- Primary Application: Computational Structural Mechanics
  - Finite Element Analysis (2-D and 3-D)
  - Problems of Importance to NASA

- Mathematical Operations:
  - Sparse Matrix Operations
    - Banded
    - Irregular
  - Eigenvalue Operations

- Secondary Applications:
  - Control Systems Analysis
  - Stability Analysis
  - Optimization/Mathematical Programming
  - Signal Processing
  - Image Processing

## APPLICATIONS

Shown on this slide are only a few of the many computationally intensive problems which may be solved on multi-microprocessors. We believe that our chordal ring approach provides substantially reduced computing time compared to other architectures due to its superiority in communications and throughput for given n, d, and k.

# USER INTERFACE

Requirements:

- **Transportability**. System must be able to run current NASA software with minimal or no modification.
- **Efficiency**. Computing tasks with high parallel content are performed as function calls to transputer network, while sequential tasks can run on front-end processor.
- **Ease-of-Use**. Tools should be provided to facilitate use of system by non-specialists.

# USER INTERFACE

Our Phase II proposal outlined the development of a multi-microprocessor computer based upon a family of expandable chordal ring architectures. However, as I've tried to stress throughout this presentation, Expert-EASE Systems places a strong emphasis on the "usabiliity" of the systems it develops. We were founded as a company in order to design, develop, and market an excellent IBM-PC based user interface to large, unwieldy codes residing on mini-computers and mainframes. Although, over the past few years our corporate direction has broadened and our technical capabilities have grown, we continue to emphasise highly user-friendly interfaces.

With this in mind, we have identified three central criteria which define the extent of the user interface required by our chordal ring based multi-microprocessor, if it is to be successfully incorporated into the computational environment at NASA/Langley, NASA/Lewis, as well as other similar establishments throughout the U.S.

These requirements are:

- Transportability. The system must be able to run all current NASA software with minimal or no modification. This requirement is derived from the NASA/Langley CSM group's desire to devote their efforts to research and not have to interrupt the flow of that work in order to utilise another computer.

- Efficiency. The chordal ring computer will operate in a "back-end" configuration with a VAX. In this configuration, specific computing tasks with high parallel content will be performed as a function calls to the back-end computer. For example, the function calls will include matrix invertion, addition, multiplication, etc. Computing tasks which are largely sequential in nature will execute on the front end VAX.

- Ease-of-Use. Tools (e.g., a large library of function calls) will be provided to facilitate the use of the system by researchers who are not parallel programming experts.

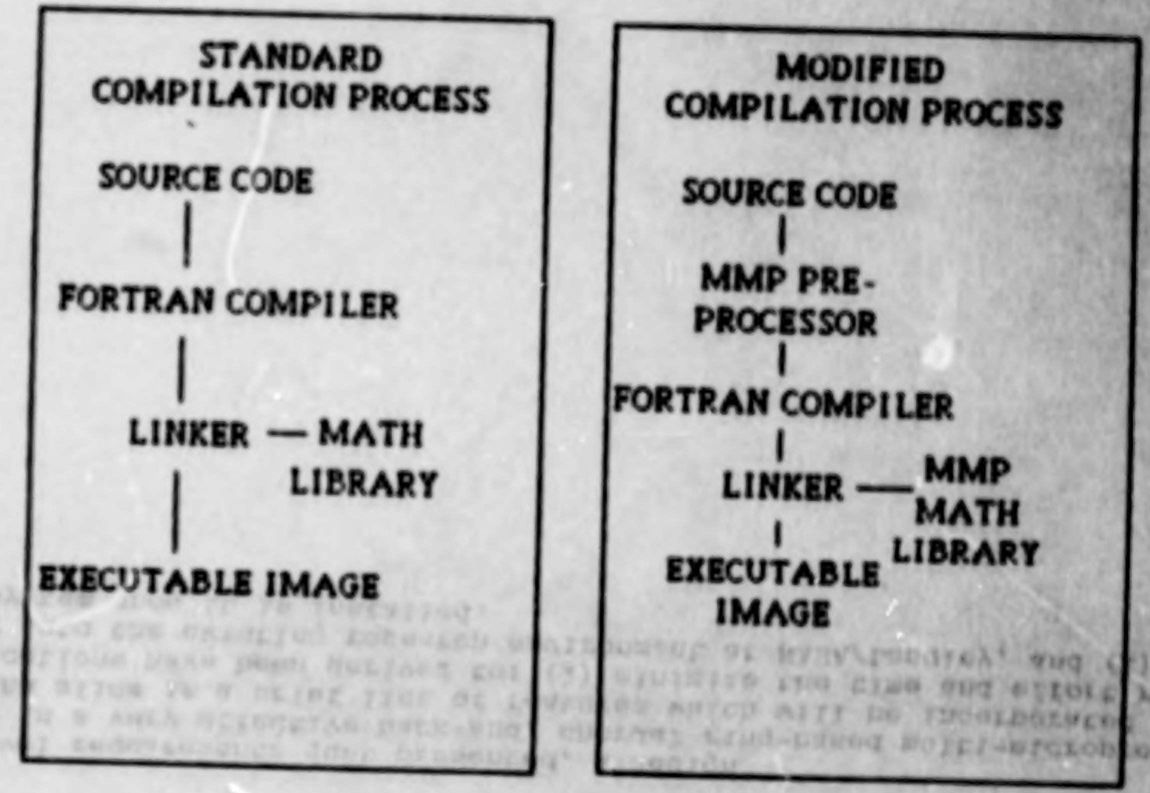
# PROPOSED SOLUTIONS

- MicroVAX platform
- MMP Pre-processor
- MMP math library
  - EES routines
  - Routines developed by NASA and the user community.
- EASE+
  - Graphics oriented front-end for input.
  - Graphics oriented post processor for display and data visualization.
- CAD interface

## PROPOSED SOLUTIONS

Based upon the high-level requirements just presented, a design approach which results in a very effective back-end, chordal ring-based multi-microprocessor has been developed. Shown on the slide is a brief list of features which will be incorporated into the computer system. These specifications have been derived to: (1) minimize the time and effort required to integrate the computer into the existing research environment at NASA/Langley, and (2) maximize the effectiveness of the system once it is installed.

# MMP PREPROCESSOR MAKES MODIFICATIONS TRANSPARENT TO THE PROGRAMMER



**STANDARD COMPILATION PROCESS**

SOURCE CODE

FORTRAN COMPILER

LINKER — MATH LIBRARY

EXECUTABLE IMAGE

**MODIFIED COMPILATION PROCESS**

SOURCE CODE

MMP PRE-PROCESSOR

FORTRAN COMPILER

LINKER — MMP MATH LIBRARY

EXECUTABLE IMAGE

## MMP PRE-PROCESSOR

This slide indicates the method by which software will be compiled for execution on the new chordal ring-based computer. It is constrasted to the standard compilation processes. The major differences between the two methods are:

- A pre-processor will convert specific multi-microprocessor calls (e.g., matrix inversion) to a form that can be handled by the FORTRAN compiler. The pre-processor will be designed so that its use requires minimal effort on the part of the researchers.

- A library of mathematical function calls will be developed. Time and cost constraints during Phase II clearly limit the number of functions that can be developed. However, we will work closely with NASA/Langley CSM Group to identify the functions of maximum utility to them. Over time, the library can be expanded by the CSM Group, Expert-EASE Systems, and other users of the system.

# DESCRIPTION OF EASE+

o Intuitive user interaction

o Graphics Database

  - Icons/Objects linked to database
  - Dynamic colors driven by any data
  - Built-in sophisticated plotting

o Full Featured Database, Menus & Forms

o Powerful Procedural Language

o Tools that Support Rapid Development

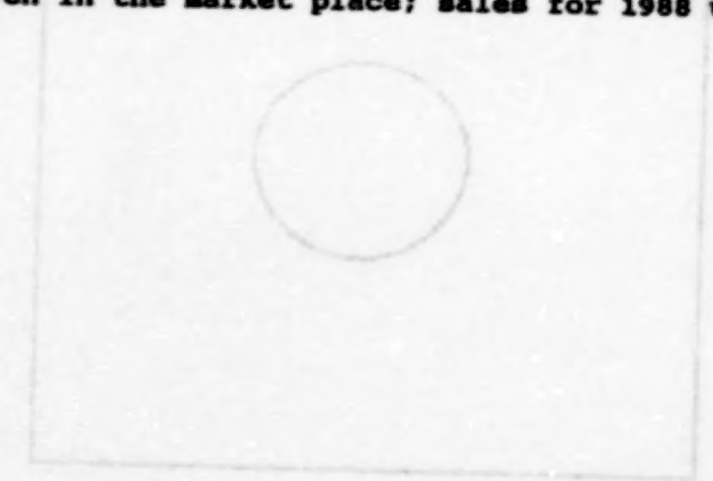
## DESCRIPTION OF EASE+

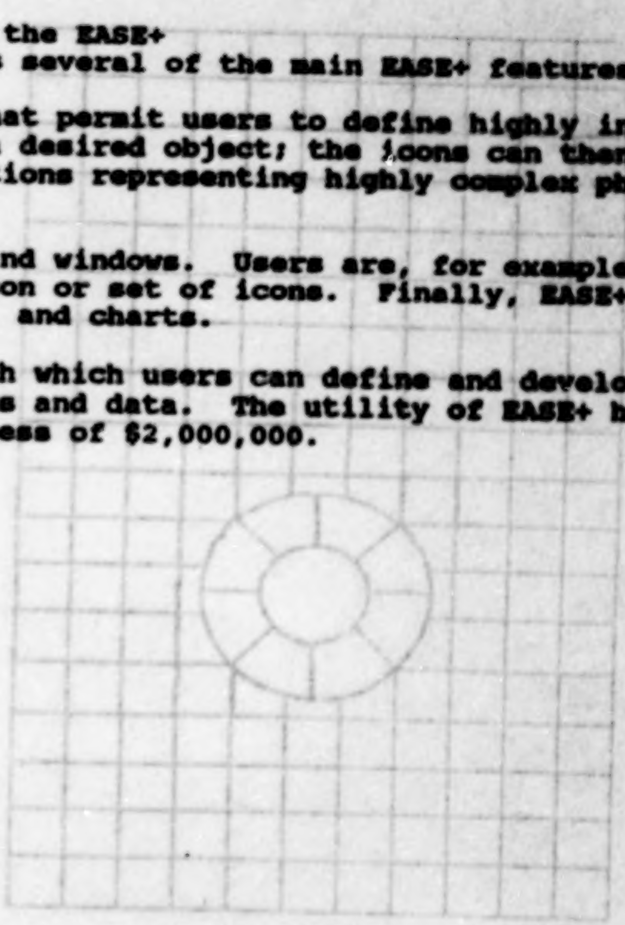
During this talk I've alluded to the general features of the EASE+ interface my company has produced. This slide highlights several of the main EASE+ features.

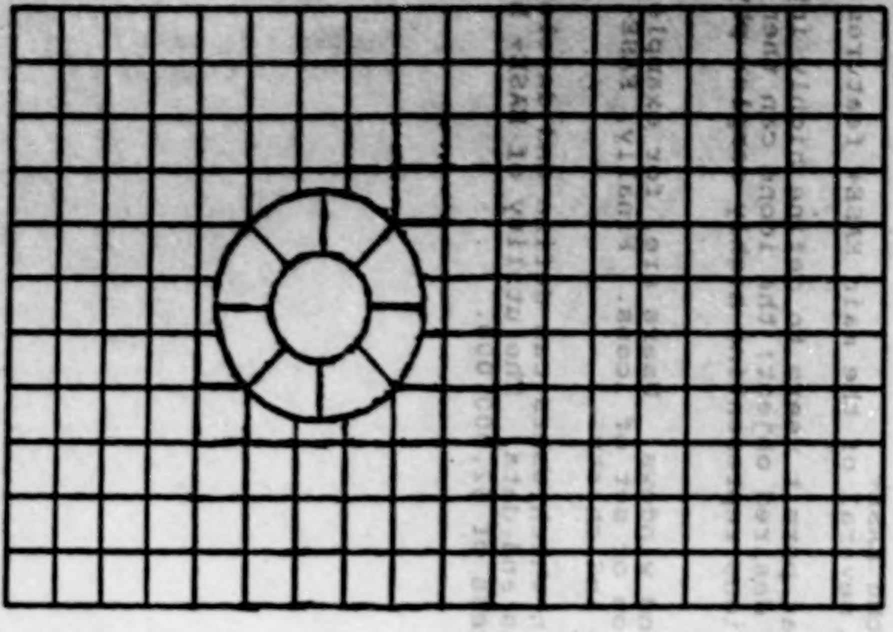
Essentially, EASE+ is a powerful set of software tools that permit users to define highly interactive graphics displays. Users can define icons to represent a desired object; the icons can then be linked to each other or to a database. In this manner, interactions representing highly complex physical processes can be displayed precisely and accurately.

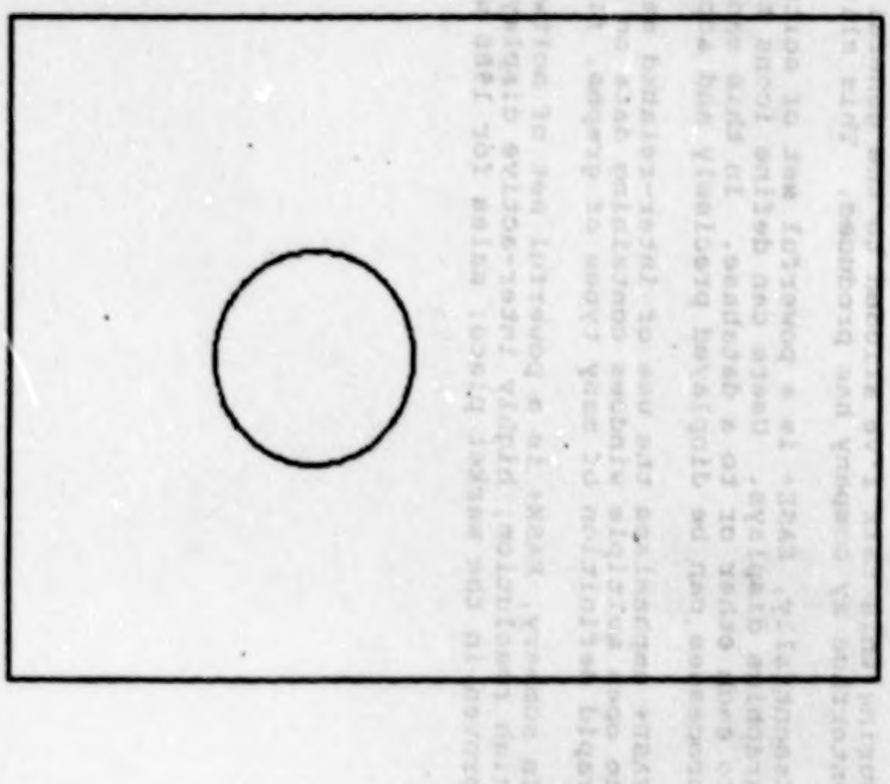
EASE+ emphasizes the use of inter-related menus, forms, and windows. Users are, for example, permitted to open multiple windows containing data on a specific icon or set of icons. Finally, EASE+ permits the rapid definition of many types of graphs, figures, plots, and charts.

In summary, EASE+ is a powerful set of software tools with which users can define and develop accurate, high resolution, highly inter-active displays of processes and data. The utility of EASE+ has been proven in the market place; sales for 1988 will be in excess of $2,000,000.







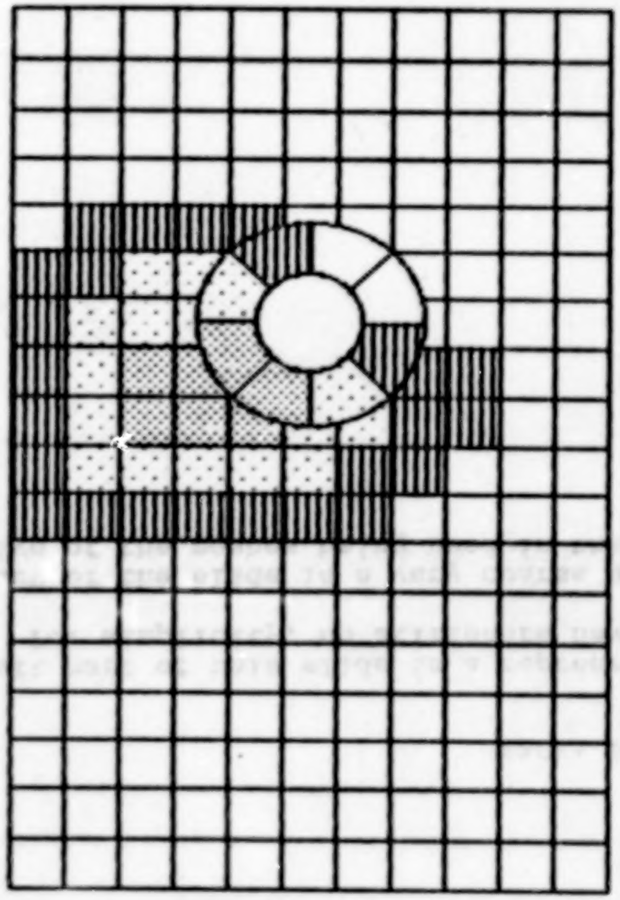


## EASE+ PRE-PROCESSING

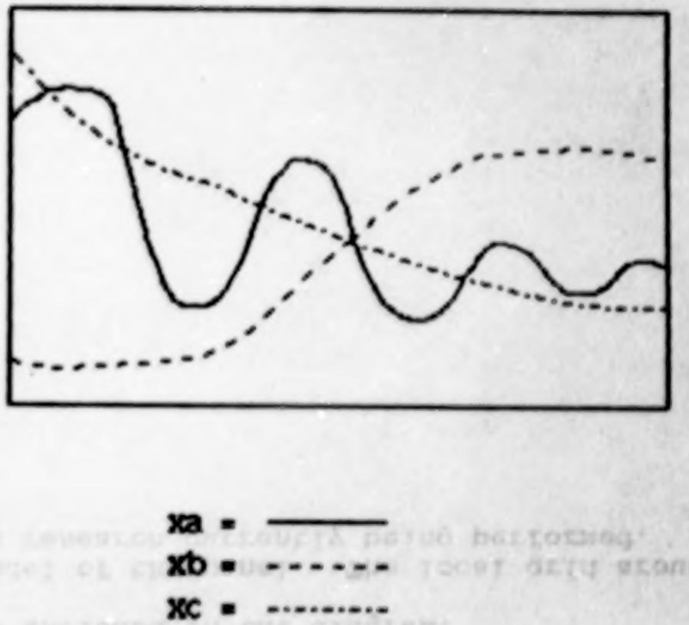
Shown on the left part of this slide is a representation of the blade panel which is of high interest to the CSM Group. For simplicity, no stiffeners have been included in the diagram.

On the right part of the slide is a very coarse grid model of the panel. The local grid around the hole is representative of the meshes being used in the plate research currently being performed.

# EASE+ POST-PROCESSING



Shaded Squares are
Regions of Critical Stress



Plots of Time
Variation Functions

# EASE+ POST-PROCESSING

Represented on the left part of this slide is the result of a stress analysis for the blade panel. For the sake of this example, only four different types of stress regions have been identified. In the chordal ring multi-microprocessor with an EASE+ user interface, this plot would be shown on a high resolution (e.g., 1280 x 1024) color CRT. Many different stress regions could be indicated by user-defined colors.

The value of EASE+ is that it would allow the user to analyze and display the stress results from different stress regions via a mouse. The user could select a particular stress region and a graph of time variations of selected functions (e.g., xa, xb, xc) in that region would be displayed in high resolution color graphics. Further, functions from different stress regions could be displayed in the same plot. Results may also be plotted via bar-graphs, charts, or other representations defined by the user.

EASE+ does not provide anything that can not be developed by an individual user using standard FORTRAN, "C" or other programming languages. However, the advantage of using EASE+ as the user interface is that it permits a higher level of system functionality to be implemented during Phase II than would otherwise be feasible. Indeed, EASE+ is the result of dozens of man-years of development time. It has evolved as the result of continual feedback from hundreds of end-users.

In summary, incorporation of EASE+ into the Phase II chordal ring multi-microprocessor would allow the computer to have an immediate impact upon the research activities of groups at both NASA/Langley and NASA/Lewis.

# SUMMARY

1. Innovative Multi-Microprocessor Computer Design Based on:

    - Inmos Transputer Chip
        - Powerful (T800 version)
        - Well-Supported

    - Modified Chordal Ring Architecture
        - Expandable
        - Low Internodal Distances
        - Low Connection Degree

    - Packaging Concept
        - Low Cost Base System (10 processors)
        - Incrementally Expandable to Over 1000 processors

2. Effective Software

    - Transportability of Existing Code
        - MICROVAX Platform
        - MMP Pre-Processor

    - Efficient Utilization of MMP
        - MMP Library
        - Function Calls

    - User-Friendly Interface
        - EASE+
        - CAD Interface
        - Visual Programming Input

ORIGINAL PAGE IS
OF POOR QUALITY

# SUMMARY

During this presentation I've attempted to summarize the proposed Phase II effort to develop a chordal ring multi-microprocessor system.

The system's hardware is based upon

- The very powerful Inmos T800 Transputer
- A modified chordal ring architecture interconnection structure that has low internodal distances and low connection degree
- A powerful packaging concept that provides a small, low cost base system of 10 Transputers, but can be expanded in a step-wise manner to over 1000 processors

The Phase II effort will emphasize the development of

- A software preprocessor to permit the utilization at existing CSM code
- A library of powerful function calls which emphasizes the manipulation of sparse matrices
- A very user-friendly interface based upon EASE+

Expert-EASE Systems is a product-oriented company with a proven track record in the U.S. and international marketplaces. Our goal is to use the Phase II project to develop a commercial multi-microprocessor which can be used in a back-end configuration with general purpose computer, such as a MicroVAX.

In summary, based upon our successful Phase I project and our proven track record in developing commercial products we are firmly convinced that we can deliver an initial, 15 Transputer system to NASA/Langley within 19 months after the start of the Phase II effort. This will be shortly followed by the marketing of the product to private businesses and government agencies as a low-cost, high performance answer to their demands for additional computing power.

# SUMMARY (Cont'd)

3. Large, Demand-Driven Markets
    - Focus on Computational Structural Mechanics
    - Incorporate Other Functions of Interest to NASA
        - Stability
        - Control
        - Optimization
    - Generalize to Other Markets
        - Signal Processing
        - Medical Imaging

N89
29780
UNCLAS

N89-29780

Seq
p 173

V312 7208

211948

32

# PARALLEL LINEAR EQUATION SOLVERS
# FOR
# FINITE ELEMENT COMPUTATIONS

Principal Investigator
James M. Ortega
University of Virginia

Gene Poole, Ph.D., 1986, Research Associate until July 1, 1987
Courtenay Vaughan, Ph.D., 1988
Andrew Cleary, Ph.D., 1988?
Brett Averick, M.S., 1987

James M. Ortega is Charles Henderson Professor and Chairman of Applied Mathematics at the University of Virginia. He is the principal investigator on NASA Grant NAG1-46 through the Computational Structural Mechanics Group and is also principal investigator on the NASA Training Grant NAG1-242.

Eugene Poole received his Ph.D. in Applied Mathematics at the University of Virginia in May 1986 and remained at the University until July 1987, partially supported under grant NAG1-46. He is now employed by Awesome Computing, Inc. and works directly with the CSM Group.

Courtenay Vaughan is a Ph.D. student in Applied Mathematics and expects to finish his degree in Spring 1988. He has been supported by NAG1-242 and also by Oak Ridge National Laboratory in the summers of 1986 and 1987.

Andrew Cleary is a Ph.D. student in Applied Mathematics with expected completion date late in 1988. He has been supported by NAG1-242 and NAG1-46 and has spent the last three summers at NASA-Langley.

Brett Averick will finish a masters degree in Applied Mathematics in Fall 1987 and plans to pursue a Ph.D. He was supported by NAG1-242 while he spent the summer of 1987 at NASA-Langley.

N89-29780

# GENERAL OBJECTIVES

Develop numerical algorithms for solution of linear and nonlinear systems of equations on parallel machines.

Apply these algorithms to problems in structural analysis, in particular, the current focus problems.

# STATUS

Choleski Banded and Profile Codes on Flex/32 for Kx = f

Conjugate Gradient and Preconditioned Conjugate Gradient Codes on Flex/32, Intel iPSC hypercube, and CRAY-2

Flex and CRAY Codes Operating in Conjunction with the CSM Testbed System

Solution for Static Displacements for Panel and Mast Focus Problems.

The overall objective of this research is to develop efficient methods for the solution of linear and nonlinear systems of equations on parallel and supercomputers, and to apply these methods to the solution of problems in structural analysis. Attention has been given so far only to linear equations.

The methods considered for the solution of the stiffness equation $Kx = f$ have been Choleski factorization and the conjugate gradient iteration with SSOR and Incomplete Choleski preconditioning. More detail on these methods will be given on subsequent slides.

These methods have been used to solve for the static displacements for the mast and panel focus problems in conjunction with the CSM testbed system based on NICE/SPAR.

# Factorization Methods on Flex/32

## Choleski Method

$$K = L\,L^T$$
$$L\,z = f,\quad L^T x = z$$

## Choleski Global Memory Code for Banded Storage

## Choleski Local Memory Code for Profile Storage

Report, December, 1986

The Choleski method first factors K into $LL^T$, the product of a lower triangular matrix and its transpose. The solution of the displacement equations $Kx = f$ is then completed by solving the triangular systems of equations $Lz = f$, $L^T x = z$.

Two versions of this method have been developed for the FLEX/32. The first uses banded storage and utilizes only global memory. The second uses profile (skyline) storage and utilizes the local memories.

Further information on these codes is given in:

A. Cleary, O. Harrar, and J. Ortega, Gaussian Elimination and Choleski Factorization on the FLEX/32. Applied Mathematics Report No. RM-86-13, University of Virginia, December, 1986.

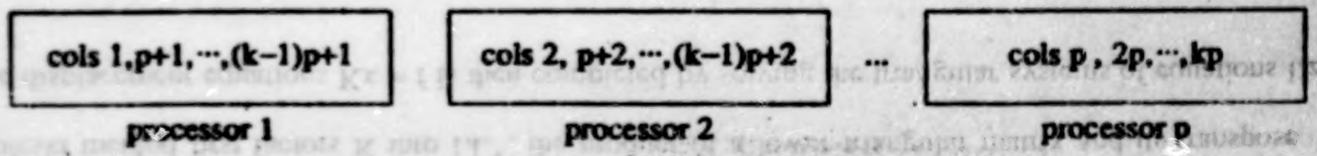
```
for k=1 to N
    $l_{kk}=a_{kk}^{1/2}$
    for s=k+1 to min(k+β,N)
      $l_{sk}=a_{sk}/l_{kk}$
    for j=k+1 to min(k+β,N)
      for i=j to min(k+β,N)
          $a_{ij}=a_{ij}-l_{ik}l_{jk}$
```

```
for k=1 to N

    cdiv (k)

    for j=k+1 to min(k+ β,N)
      cmod(j,k)
```

**kji Choleski Factorization**

```
if (column 1 is assigned to this processor)
    cdiv(1)
    mark column 1 done
for k=1 to N−1
    wait for column k to be done
    if (column k+1 is assigned to this processor)
        cmod(k+1,k)
        cdiv(k+1)
        mark column k+1 done
    for j = k+2 to min(k+β,N)
        if (column j is assigned to this processor)
            cmod(j,k)
```

**Parallel Choleski Factorization With Compute-Ahead**


cols 1,p+1,···,(k−1)p+1
cols 2, p+2,···,(k−1)p+2
...
cols p , 2p,···,kp
processor 1
processor 2
processor p

**Wrapped Interleaved Column Storage**

ORIGINAL PAGE IS
OF POOR QUALITY

The first figure shows a basic Choleski factorization code using the so-called kji form (L is computed by columns using immediate updates). For simplicity, banded storage with bandwidth β is used in this code. To the right of the first code is shown the same code in a short-hand version: cdiv(k) forms the first column of L and cmod(j,k) modifies the jth column of K using the kth column of L.

The second figure illustrates the code for each processor on the FLEX/32. Synchronization is achieved by marking columns of L "done" when they are computed. At the kth stage, the (k+1)st column in L is first computed and marked done so as to make this column available to other processors as soon as possible; this is the "compute-ahead" since the usual algorithm would compute the (k+l)st column only after the kth stage had been computed.

The third figure shows the storage assignment of the columns of K to the local memories. This interleaving of the columns of K has been observed by several investigators to give good load balancing in a local memory parallel system.

### Choleski on the Flex/32

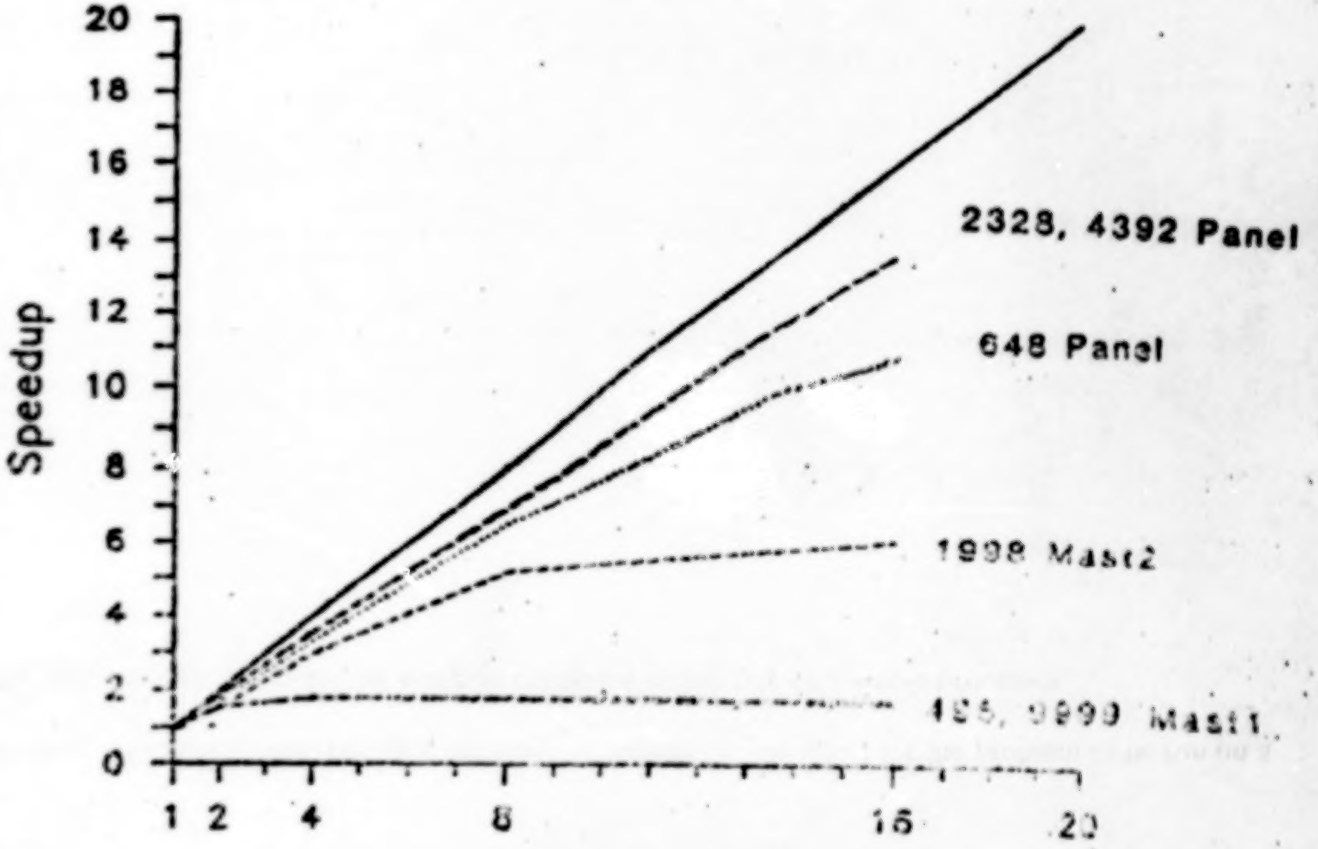
| Problem | N | β | Storage | p | Time(*secs.*) | Mflops | Speed Up* |
|---|---|---|---|---|---|---|---|
| Panel | 648 | 118 | 34978 | serial | 63.9 | .0418 | |
| | | | | 1 | 65.0 | .0411 | .98 |
| | | | | 2 | 33.3 | .080 | 1.91 |
| | | | | 4 | 17.5 | .153 | 3.66 |
| | | | | 8 | 9.70 | .276 | 6.60 |
| | | | | 16 | 6.30 | .424 | 10.14 |
| Panel | 2328 | 240 | 243126 | serial | 735 | .0439 | |
| | | | | 1 | 744 | .0434 | .99 |
| | | | | 2 | 375 | .0861 | 1.96 |
| | | | | 4 | 195 | .166 | 3.78 |
| | | | | 8 | 101 | .320 | 7.29 |
| | | | | 16 | 55.5 | .582 | 13.26 |
| Panel | 4392 | 260 | 501853 | serial | | | |
| | | | | 1 | | | |
| | | | | 2 | 839 | .0858 | 1.95 |
| | | | | 4 | 432 | .167 | 3.80 |
| | | | | 8 | 223 | .323 | 7.36 |
| | | | | 16 | 120 | .600 | 13.67 |

* Speed Up for problems too large for single processor are computed using estimated Mflop rate on one processor

ORIGINAL PAGE IS
OF POOR QUALITY

This table shows running times for the parallel Choleski code for the panel focus problem on the FLEX/32. Timings are given for 1, 2, 4, 8 and 16 processors. The corresponding speedups and mflop rates are also given. The speed-ups are calculated using a serial code. For comparison, times are also given for the parallel code on a single processor.

For the largest problem, 4392 unknowns, the local memory (4 Mbytes) is too small for the problem to be run on a single processor, and the speedups are computed by using an estimated mflop rate on a single processor.



Parallel Choleski Mast and Panel Results

ORIGINAL PAGE IS
OF POOR QUALITY

This figure plots the speedups for the panel focus problems as well as two different mast problems. The first mast problem has a small bandwidth (15) and very poor speedups are obtained. The second mast problem has a bandwidth of about 50 and the speed-ups are better. The panel problems have much larger bandwidths, as given in the table, and the speed-ups are much better. In general, the speedups given by this Choleski code are primarily a function of the bandwidth and not of the number of unknowns. This is illustrated, in particular, for the first mast problem and the smallest panel problem, both of which have almost the same number of unknowns.

# Conjugate Gradient Iteration

FLEX/32

- **Mast and Panel Focus Problems**
- **Incomplete Choleski Preconditioning Using FORCE**
- **SSOR Polynomial Preconditioning**

CRAY-2 (Ames)

- **CG Running on Panel Focus Problem**
- **Preconditioning Later**

Intel iPSC Hypercube (Oak Ridge)

- **CG Running on Panel Focus Problem**
- **Preconditioning Later**

ORIGINAL PAGE IS
OF POOR QUALITY

The second type of method is iterative, the conjugate gradient method with two different preconditioners: SSOR polynomial and incomplete Choleski factorization. These methods have been used to solve both the panel and mast focus problems on the FLEX/32. The incomplete Choleski codes for the FLEX/32 use the FORCE package developed by H. Jordan. The conjugate gradient method has also been used for the panel focus problem on an Intel iSPC 64-processor hypercube at Oak Ridge National Laboratory, and on the CRAY-2 at NASA-Ames; preconditioning for these codes is under development.

# Preconditioned Conjugate Gradient Code

```
Choose $x^0$.  Set $r^0 = f - Kx^0$.  Solve $M\tilde{r}^0 = r^0$.  Set $p^0 = \tilde{r}^0$.
For $k = 0, 1,$
    one dimensional minimization  $\left\{ \begin{array}{l} \alpha_k = -(\tilde{r}^k, r^k)/(p^k, Kp^k) \\ x^{k+1} = x^k - \alpha_k p^k \end{array} \right.$
    update residual  $r^{k+1} = r^k + \alpha_k Kp^k$
    Test for convergence
    preconditioning  Solve $M\tilde{r}^{k+1} = r^{k+1}$
    new conjugate direction  $\left\{ \begin{array}{l} \beta_k = (\tilde{r}^{k+1}, r^{k+1})/(\tilde{r}^k, r^k) \\ p^{k+1} = \tilde{r}^{k+1} + \beta_k p^k \end{array} \right.$
```

Note: **Key parts are $Kp^k$ and preconditioning**

ORIGINAL PAGE IS
OF POOR QUALITY

A code is given for the preconditioned conjugate gradient method. ( , ) is the inner product of two vectors. The first two statements compute the next iterate $\mathbf{x}^{k+1}$ by minimizing the quadratic function $\mathbf{x}^T K \mathbf{x} - 2\mathbf{x}^T \mathbf{f}$ along the line $\mathbf{x}^k - \alpha \mathbf{p}^k$. The residual at $\mathbf{x}^{k+1}$ is then computed and there is a test for convergence. This test can be based on $(\mathbf{r}^{k+1}, \mathbf{r}^{k+1})$, or other tests can be used. The next step carries out the preconditioning by solving a system of equations with coefficient matrix M. Finally, the last two statements compute the next direction vector $\mathbf{p}^{k+1}$.

The potentially time-consuming parts of this process are the matrix-vector multiply $K\mathbf{p}^k$ and the preconditioning.

# Preconditioners

## Incomplete Choleski

$K = LL^T - R, \quad M = LL^T$

No Fill: L has same sparsity as K

$M\tilde{r} = r$: $Lz = r, \quad L^T\tilde{r} = z$
Done each step of iteration

## SSOR

An SOR iteration followed by an SOR iteration in reverse direction

m-step SSOR: m SSOR iterations

Polynomial SSOR: combine the m SSOR iterations in optimal way

Note: Multicoloring is used to parallelize SOR

ORIGINAL PAGE IS
OF POOR QUALITY

Two preconditioners are used in conjunction with the conjugate gradient iteration. The first is incomplete Choleski factorization, in which the simplest strategy is no-fill: the factor L has a non-zero only if the corresponding position of K is non-zero. The factorization itself is done only once at the beginning of the iteration and the key point is the forward and back substitutions $Lz = r$ and $L^T \bar{r} = z$, which are done at each iteration. These are achieved by a sparse form of the column sweep algorithm with L stored by interleaved rows.

The second preconditioner is based on the SSOR iteration. This is a combination of an SOR iteration followed by another SOR iteration in the reverse direction. More than one SSOR iteration can be taken, and these SSOR steps can be weighted to give what is called SSOR polynomial preconditioning. Multicoloring of the nodes is used to parallelize the SOR iteration.

## SSOR Preconditioned Conjugate Gradient

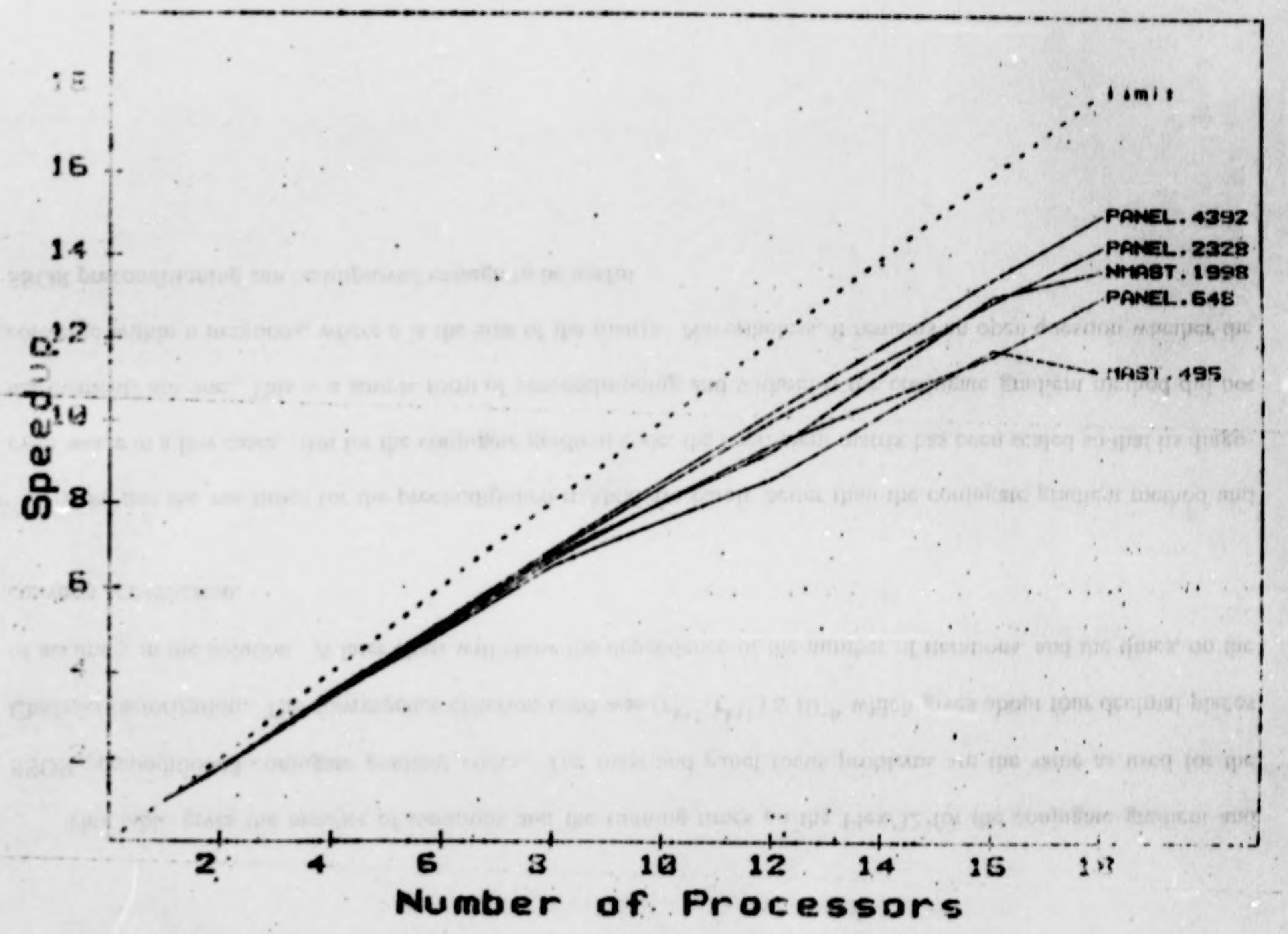
| | Time (seconds) | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | MAST.495 | | NMAST.1998 | | PANEL.648 | | PANEL.2328 | | PANEL.4392 | |
| | CG | PCG | CG | PCG | CG | PCG | CG | PCG | CG | PCG |
| iter | 445 | 205 | 390 | 159 | 299 | 126 | 1152 | 422 | 1238 | 487 |
| 1 | 144.2 | 147.5 | 683.5 | 629.6 | 154.8 | 149.4 | 2522 | 2066 | 5416 | 4743 |
| 2 | 82.2 | 82.8 | 379.1 | 371.7 | 87.9 | 84.5 | 1407 | 1138 | 3016 | 2626 |
| 4 | 42.1 | 43.3 | 197.1 | 211.3 | 45.2 | 45.1 | 717 | 576 | 1530 | 1315 |
| 8 | 22.2 | 23.4 | 101.6 | 110.5 | 23.4 | 25.1 | 372 | 304 | 789 | 690 |
| 12 | 16.7 | 16.8 | 73.3 | 74.7 | 16.3 | 19.8 | 251 | 215 | 528 | 468 |
| 16 | 12.3 | 13.5 | 52.3 | 58.6 | 13.3 | 17.8 | 195 | 173 | 401 | 370 |
| 18 | 12.8 | 12.3 | 49.9 | 46.8 | 11.8 | 15.2 | 177 | 157 | 360 | 322 |

ORIGINAL PAGE IS
OF POOR QUALITY

This table gives the number of iterations and the running times on the Flex/32 for the conjugate gradient and SSOR preconditioned conjugate gradient codes. The mast and panel focus problems are the same as used for the Choleski factorization. The convergence criterion used was $(r^{k+1}, r^{k+1}) \leq 10^{-6}$ which gives about four decimal places of accuracy in the solution. A later chart will show the dependence of the number of iterations, and the times, on the convergence criterion.

Note that the run times for the preconditioned method are barely better than the conjugate gradient method and even worse in a few cases. But for the conjugate gradient code, the coefficient matrix has been scaled so that its diagonal elements are one. This is a simple form of preconditioning and without it the conjugate gradient method did not converge within n iterations, where n is the size of the matrix. Nevertheless, it remains an open question whether the SSOR preconditioning can be improved enough to be useful.

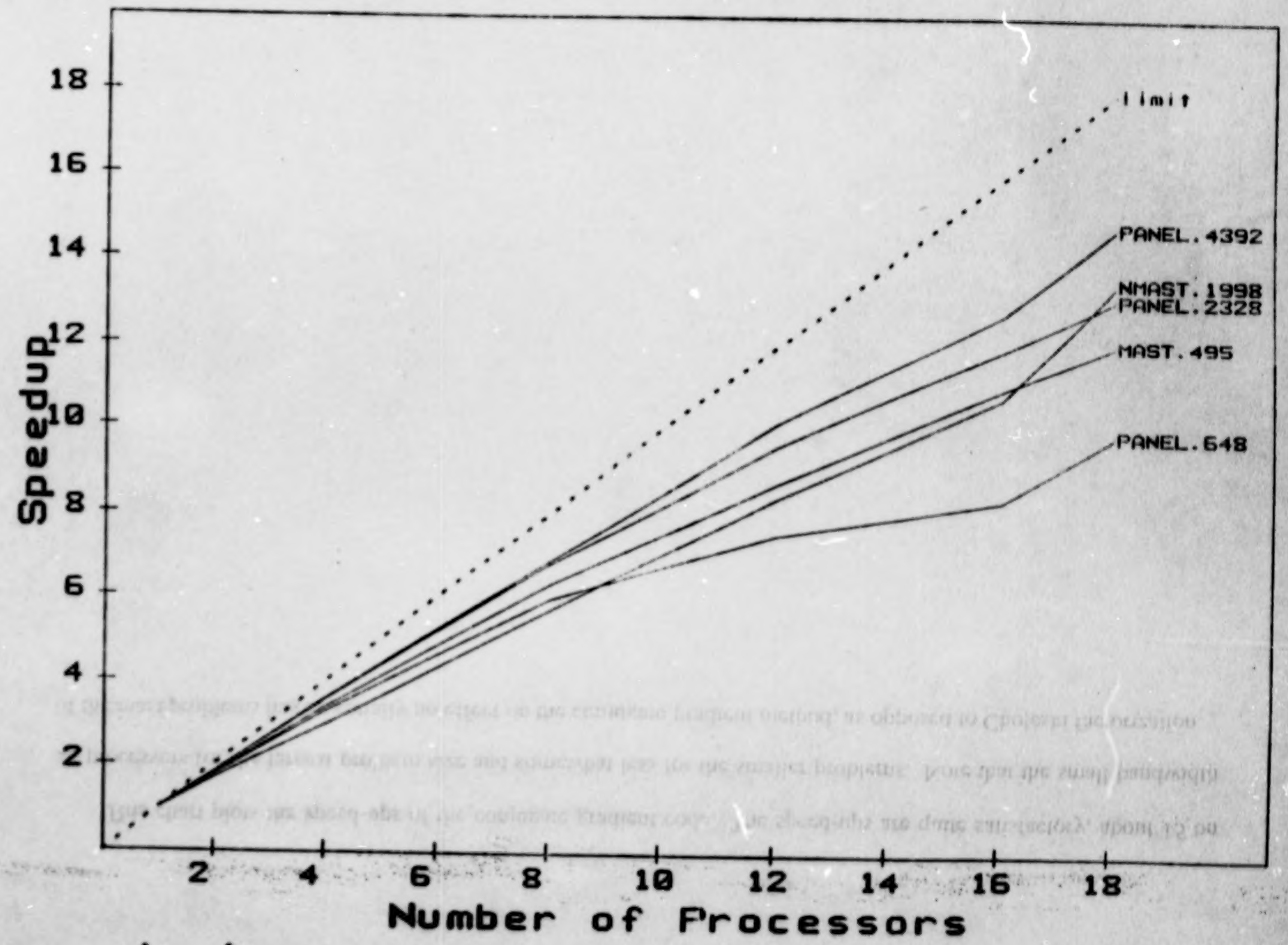
## Speedup CG Flex



ORIGINAL PAGE IS
OF POOR QUALITY

This chart plots the speed-ups of the conjugate gradient code. The speed-ups are quite satisfactory, about 15 on 18 processors for the largest problem size and somewhat less for the smaller problems. Note that the small bandwidth of the mast problems has essentially no effect on the conjugate gradient method, as opposed to Choleski factorization.

# Speedup PCG Flex



ORIGINAL PAGE IS
OF POOR QUALITY

This chart shows the speed-ups of the SSOR polynomial preconditioned conjugate gradient method. These speed-ups are a little worse than for the conjugate gradient method but still satisfactory.

**Comparison of Choleski and Preconditioned Conjugate Gradient (Seconds)**

| Panel | p | Choleski | PCG |
|---|---|---|---|
| 648 | 1 | 64 | 150 |
| | 16 | 6.3 | 18 |
| 2328 | 1 | 744 | 2066 |
| | 16 | 56 | 173 |
| 4392 | 2 | 839 | 2626 |
| | 16 | 110 | 370 |

This chart compares the run times in seconds on the FLEX/32 of the Choleski factorization and preconditioned conjugate gradient methods. Times are given for three sizes of the panel focus problem and for 1 and 16 processors (except for the largest problem which Choleski could not run on a single processor).

The Choleski method has a clear advantage on this problem.

## Conjugate Gradient for PANEL.648

ε vs. error

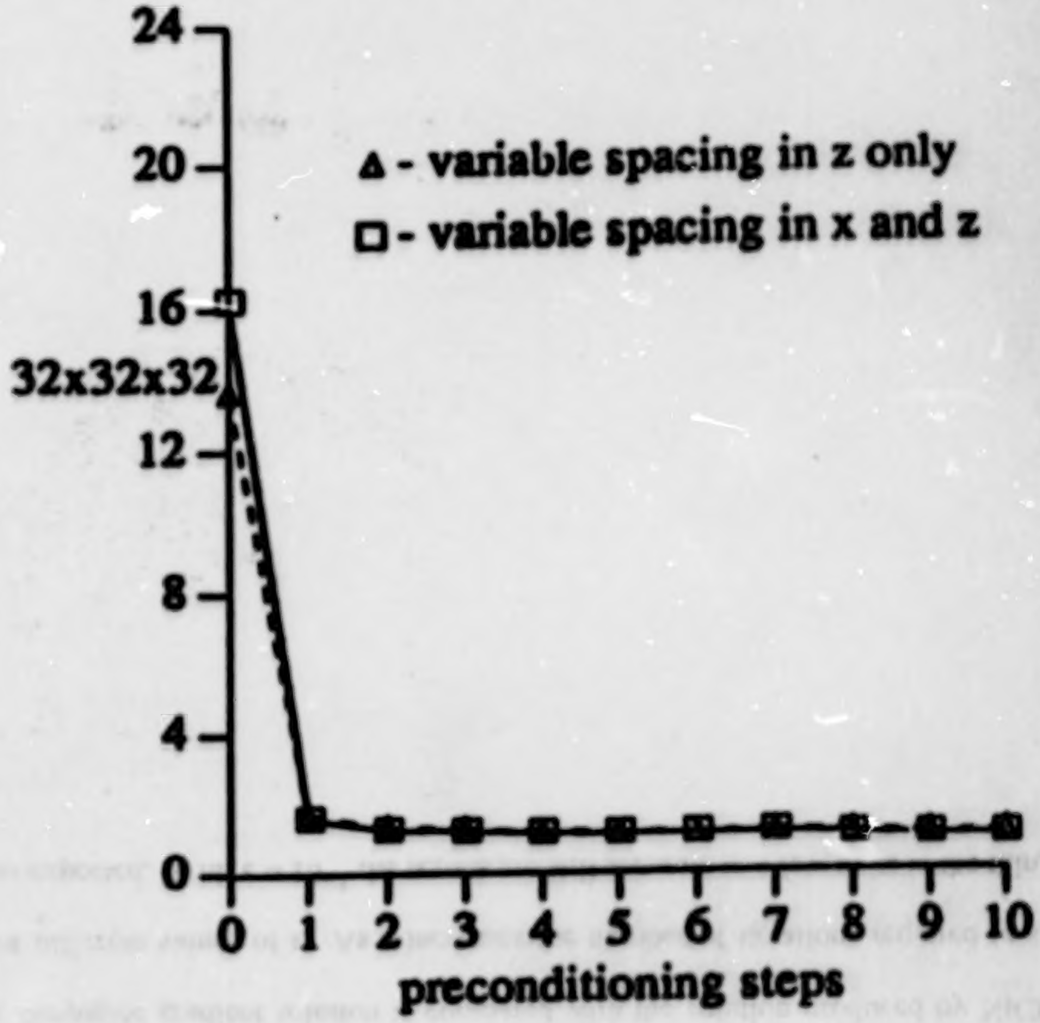
$(r^{k+1}, r^{k+1}) \leq \varepsilon$

maximum answer 0.216670

| ε | maximum error | iterations |
|---|---|---|
| $10^{-6}$ | 0.000056 | 299 |
| $10^{-5}$ | 0.000697 | 268 |
| $10^{-4}$ | 0.001966 | 230 |
| $10^{-3}$ | 0.003683 | 188 |

The running times for an iterative method depend critically upon the convergence criterion. This table shows the results of varying the parameter ε in the convergence test $(r^{k+1}, r^{k+1}) \le \varepsilon$ for the conjugate gradient method.

The approximate conjugate gradient solution is compared with the solution produced by NICE/SPAR to obtain the maximum error for different values of ε. As ε increases the number of iterations required to satisfy the convergence test decreases, as expected. With $\varepsilon = 10^{-3}$ the results are still accurate to a few units in the third decimal place.



SSOR Preconditioned Conjugate Gradient for $\nabla (a \nabla u) = f$

Time on Cyber 205

The chart shows what can be achieved by preconditioning. The problem is a three-dimensioned Poisson-type equation of the form $\nabla (a \nabla u) = f$, discretized by finite differences. For a 32 × 32 × 32 cube, one step of SSOR preconditioning reduces the time by almost a factor of 10. For reasons which are not yet clear, this problem is very suitable for SSOR preconditioning whereas the focus problems are not.

## Conjugate Gradient on CRAY-2

**Panel 648**

Conjugate Gradient: .26 seconds

NICE/SPAR: 2.2 seconds

**Panel 2328**

Conjugate Gradient: 3.5 seconds

NICE/SPAR: 18.9 seconds

Notes: NICE/SPAR Times Using Minimum Degree Ordering

Conjugate Gradient Convergence: $(r^{k+1}, r^{k+1}) \leq 10^{-6}$

This chart gives running times on a single processor of the CRAY-2 at NASA-Ames for two sizes of the panel problem. For comparison, times are also given for the solution phase of NICE/SPAR. The NICE/SPAR program uses the minimum degree ordering of the unknowns, which is favorable for the algorithms in NICE/SPAR. Note, however, that NICE/SPAR has not been optimized for the CRAY-2.

# Summary and Conclusions

Choleski factorization gives good speedups if bandwidth sufficiently large but poor for small bandwidth.

Conjugate gradient gives good speedups independent of bandwidth.

Choleski factorization almost three times faster than preconditioned conjugate gradient on panel focus problem. But this depends on convergence test used for conjugate gradient.

N89
29781

UNCLAS

TN 840743
ØA789476
N89-29781
58-39
368
211949

# Algorithms and Software for Solving Finite Element Equations on Serial and Parallel Architectures

Alan George
Dept. of Computer Science, University of Tennessee
and
Mathematical Sciences Section, Oak Ridge National Laboratory

## Abstract

Over the past 15 years numerous new techniques have been developed for solving systems of equations and eigenvalue problems arising in finite element computations. A package called SPARSPAK has been developed by the author and his co-workers which exploits these new methods. The broad objectives of this research project is to incorporate some of this software in the CSM testbed, and to extend the techniques for use on multiprocessor architectures. ~~The work is being supported by NASA grant NAG-1-803.~~

N89-29781

## General Objectives

- Develop and test algorithms and software for solving finite element systems of equations.
- Install software in the CSM testbed, and conduct comparisons of new algorithms with those already in the testbed.
- Extend or adapt the algorithms for use on parallel architectures.

# General Objectives

The general objective of the project is to perform research into sparse matrix techniques for solving systems of equations and eigenvalue problems arising in finite element computations. The research is to be done in the context of the CSM testbed employed by the Structural Mechanics Branch at the Langley Research Center. In addition to providing new and hopefully better capabilities for users of the testbed, the automatic generation of realistic structural mechanics problems will provide a good environment in which to develop new algorithms for solving sparse systems.

The proposed research is to be conducted over a period of two years. In the first year of the grant, which began Sept. 1, 1987, existing sparse matrix software developed by the principal investigator and his co-workers will be modified and installed in the CSM testbed, and performance comparisons with existing sparse matrix techniques already in the testbed will be conducted. In addition, the application of some recently developed sparse matrix techniques for handling indefinite problems will be explored in connection with solving eigenvalue problems generated by the testbed.

In the second year of the proposed research program, the focus of the investigation will be on extending the basic techniques to exploit multiprocessor architectures, again in the context of the CSM testbed. Ideas and software already developed or in development are to be adapted and incorporated into the testbed, and performance studies will be conducted.

## Specific Tasks

- Modify and install some software from the sparse matrix package SPARSPAK in the CSM testbed. Status: testbed installed on SUN workstation.
- Compare performance with existing sparse matrix techniques already in the testbed.
- Application of some new techniques for solving *indefinite* problems to the solution of eigenvalue problems generated by the testbed. Status: basic algorithm and software developed.
- Extend the methods employed in SPARSPAK for use on multiprocessors.
- Compare performance with parallel methods already available in the testbed.
- Develop new algorithms and software, particularly in the area of automatic mapping of tasks to processors.

## Specific Tasks

An initial objective will be to compare the performance of state-of-the-art techniques for solving sparse systems with those that are currently available in the CSM testbed. Thus, one of the early tasks will be to become familiar with the structure of the testbed, and to install some or all of the SPARSPAK package in the testbed. This will allow performance studies to be conducted. To date, the CSM testbed has been installed on a network of SUN workstations at the University of Tennessee, and the demonstration problems have been run successfully. Studies on how best to integrate SPARSPAK into the testbed are currently in progress. Installation of the testbed uncovered a few minor bugs in the install procedure, but apart from that, the process of installing the testbed in the UNIX environment is essentially automatic and very straightforward.

The testbed requires the solution of non-positive definite systems in connection with solving the eigenvalue problem via inverse iteration. Recently the principal investigator and his colleagues have developed a so-called static-storage partial pivoting scheme for solving indefinite sparse matrix equations. The intention is to incorporate this new algorithm into the testbed, and conduct comparisons with those already available in the testbed. The basic software to implement the algorithm has been completed and tested. The next step is to incorporate it into the testbed in an appropriate way.

A final objective of the proposed research is to extend the methods employed in SPARSPAK for use on multiprocessors. Considerable research and development in this direction has already been done, Some of the algorithms and codes developed by the principal investigator and his colleagues will be installed in the CSM testbed, and performance studies will be conducted to allow comparisons with those already available as part of the testbed. Other algorithms and software may have to be developed.

# Symmetric Positive Definite Systems

When $A$ is s.p.d., solving $Ax = b$ typically involves four distinct steps:

1. (Ordering) Find a good ordering for $A$ . That is, a permutation matrix $P$ so that $PAP^T$ has a sparse Cholesky factor $L$ .

2. (Symbolic factorization) Determine the structure of $L$, and set up a data structure for this factor.

3. (Numerical factorization) Place the elements of $A$ into the data structure, and then compute $L$.

4. (Triangular solution) Using $L$ , solve the triangular systems $Ly = Pb, L^T z = y$, and then set $x = P^T z$ .

SPARSPAK contains state-of-the-art algorithms for dealing with steps 1–4.

# Symmetric Positive Definite Systems

One of the key advantages of symmetric positive definite matrices is that Gaussian elimination applied to them does not require interchanges (pivoting) to maintain numerical stability. Since $PAP^T$ is also symmetric and positive definite for any permutation matrix $P$, this means we can choose to reorder $A$ symmetrically *without regard to numerical stability* and *before the actual numerical factorization begins.*

These options, which are normally not available to us when $A$ is a general indefinite matrix, have enormous practical implications. Since the ordering can be determined before the factorization begins, the locations of the fill suffered during the factorization can also be determined. Thus, the data structure used to store $L$ can be constructed prior to the actual numerical factorization, and spaces for fill components can be reserved. The use of a static data structure allows the data structure to *very* efficient. On average, usually there is *less that one* item of overhead storage (pointer, subscript, etc.) for each component of the matrix $L$.

The computation then proceeds with the storage structure remaining *static* (unaltered). Thus, the three problems of i) finding a suitable ordering, ii) setting up the appropriate storage scheme, and iii) the actual numerical computation, can be isolated as separate objects of study, as well as separate computer software modules.

# SPARSPAK - Design Considerations

- Computer programs for solving sparse systems of linear equations typically involve fairly complicated data structures and storage management.
- The unconventional data structures usually mean that subroutines have long argument lists, involving parameters which are of no interest to the user.
- In most cases the user of such programs simply wants to solve the problem, and should not have to understand how the storage management is done, or how the matrix components are actually stored.
- The user should be insulated from these complications, but should still be able to use the package in a variety of ways.
- Ideally, the only information the package should require is that which the user must know anyway.

# SPARSPAK - Design Considerations

Computer subroutines for solving dense matrix problems involve conventional numerical data types such as one- or two-dimensional arrays of floating point numbers, which are already available in the programming languages normally used for scientific computation. The storage requirement is known as soon as the dimension of the problem is prescribed, and the number of parameters to such procedures is usually quite modest. In addition, the number of subroutines involved in solving a problem is only one or two. Thus, the "intellectual overhead" in learning how to use the subroutines is quite small.

Unfortunately, very little of this holds for subroutines which implement algorithms for solving sparse systems. The algorithms are relatively complicated, and their implementations typically involve a substantial number of subroutines. The data structures are sufficiently unconventional that they are not provided as standard data types, so subroutines usually have long parameter lists, most of which have no meaning to the user unless he or she cares to know how the data are stored. Finally, the amount of storage is usually unpredictable until at least part of the overall computation has been completed, which complicates the management of computer storage.

## SPARSPAK - Features

- Contains implementations of state-of-the-art algorithms.
- Has a friendly *user interface*, which is a layer of software between the user and the numerous subroutines which implement the four phases of the solution procedure.
- The interface provides storage management services, sequencing control, checkpoint and restart facilities, and insulation from long subroutine argument lists.

# SPARSPAK - Features

SPARSPAK was designed to address the complications outlined on the previous slide. The package contains implementations of state-of-the-art algorithms, but its novel and unique feature is its *user interface*, which is a layer of software between the user, who has a sparse problem to solve, and the numerous subroutines which implement the four phases of the solution procedure mentioned above. This layer of software provides storage management services, insulates the user from the complicated data structures used by the subroutines, and provides a convenient means of communication between the user and the subroutines. The user is required to know very little more than what he or she must know anyway about the problem to be solved. The interface also provides sequencing control, so that subroutines are called in the correct order, and convenient checkpoint and restart facilities. The latter capability, along with the modular design of the package, allows the user to avoid re-doing parts of the computation if it is aborted after part of it has been successfully completed.

# SPARSPAK - Structure of the Package

The user's program and the package interact as follows:

1. (Structure Input) The user supplies the nonzero structure of $A$.
2. (Order) The package reorders the original problem, (finds a permutation $P$), and allocates storage for the triangular factorization of $PAP^T$.
3. (Numerical Input) The user supplies the numerical values for the matrix $A$ to the package.
4. (Factor) The package computes the triangular factors of $PAP^T$.
5. (Numerical Input) The user supplies numerical values for $b$. (This step may come before Step 4, and may be intermixed with Step 3.)
6. (Solve) The package computes the solution $x$, using $L$, $P$ and $b$.

# SPARSPAK - Structure of the Package

In step one, the user communicates the structure of the matrix to the package by calling one or more simple interface routines. A typical program statement would be *CALL INIJ(I,J)*, which would tell the package that there is nonzero element in position $(i,j)$. There are also routines that can be used to communicate the structure corresponding to an entire element stiffness matrix in one call. After the structure is input, a single statement such as *CALL ORDER* invokes an ordering routine (there are several choices), and sets up the data structure.

The user then provides the numerical values to the package by executing a call to one or several routines. These routines allow assembly of the element matrices to be done in a perfectly natural way. The package "knows" when to initialize the data structure, and after that, all input to the package is additive. A typical program statement for numerical input would be *CALL INAIJ(I,J,AIJ)*.

After the numerical input is complete, a single call to a subroutine named *SOLVE* initiates the factorization. If the right hand side has not been input, only the factorization will be performed. If it has, then both factorization and solve will be done. The right hand side is supplied to the package using routines similar to those for inputting the matrix components.

# SPARSPAK - Installation in the Testbed

Strategies, issues:

- Leave SPARSPAK more-or-less intact, and implement a single processor that performs the functions of TOPO, K, INV, and SSOL. Inflexible with respect to nonlinear analysis and the eigenvalue problem.
- Decompose SPARSPAK into a number of processors, which serve as alternatives to JSEQ or RSEQ, K, INV and SSOL. More flexible.
- In order to ensure that existing processors can be used in concert with new SPARSPAK-derived ones, it is important that the existing data structures be preserved.

# SPARSPAK - Installation in the Testbed

One possibility is to leave SPARSPAK more or less intact. This would imply creating a single processor which would first obtain topological information about the structure from the data base, along with constraint information indicating which variables are constrained. This would then allow SPARSPAK to perform the reordering, set up the data structure for the overall stiffness matrix, and allocate storage for it. The processor would then again access the data base to obtain the element stiffness matrices and applied loads. Using constraint information, the overall assembly could then be performed using the basic facilities that are in SPARSPAK. Once the assembly is complete, factorization and solution can be done.

Alternatively, SPARSPAK could be decomposed into several processors which would serve as alternatives to JSEQ (RSEQ), K, INV, and SSOL.

These alternatives and others are currently being explored with technical personnel at LARC. The first alternative is quite inflexible, and would not lend itself well to many current applications of the testbed. The second provides more flexibility, but if the new processors derived from SPARSPAK are to be used in concert with existing ones, the data/file structures that provide the information exchange among the processors must be maintained.

RSEQ already provides a reordering capability, but it reorders the problem without knowledge of constraints. SPARSPAK will be most efficient if it is allowed to reorder and assemble the matrix *after* the constraints have been applied. One objective is to determine whether exploiting the effect of the constraints is worth the effort. The advantage of the current approach used by RSEQ is that many constraint sets can be considered with only one pass through RSEQ.

# Detecting Parallelism in Sparse Matrix Computation

- **Objectives:**
  - **preserve sparsity**
  - **low arithmetic operation count**
  - **high parallelism**
  - **low communication**
- **These objectives turn out to be complementary.**
- **Good (serial) orderings are also good for parallel computation, or can be rearranged so that they are good.**
- ***Elimination trees* are useful in this analysis.**

# Detecting Parallelism in Sparse Matrix Computation

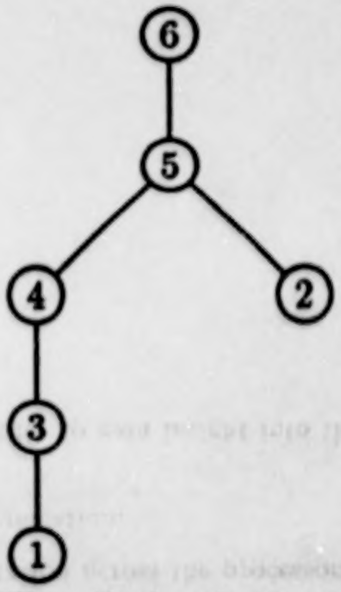
Ideally, we would like to choose an ordering for the matrix $A$ which achieves a number of objectives. First, just as in the use of serial machines, we would like to preserve sparsity and obtain a low arithmetic operation count. In addition, the ordering should allow a high degree of parallelism, and allow the distribution of the computation across the processors in a way that allows the parallelism to be exploited without requiring an inordinate amount of communication.

Fortunately, these objectives turn out to be mutually complementary. In order to gain insight into this problem, it is useful to introduce the notion of elimination trees for sparse Cholesky factors.

# Detecting Parallelism - elimination trees

$$L = \begin{pmatrix} \times & & & & & \\ & \times & & & & \\ \times & & \times & & & \\ & & \times & \times & & \\ \times & \times & \times & \times & \times & \\ & \times & & \times & \times & \times \end{pmatrix}$$

Structure of a Cholesky factor



The elimination tree

# Detecting Parallelism - elimination trees

Consider the structure of the Cholesky factor $L$. For each column $j \le n$, if column $j$ has off-diagonal nonzeros, define $\gamma[j]$ by

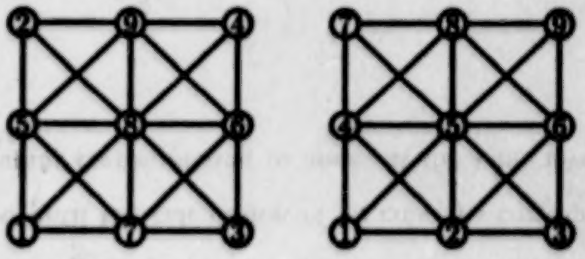
$$\gamma[j] = \min\{i \mid l_{ij} \neq 0, i > j\};$$

that is, $\gamma[j]$ is the row subscript of the first off-diagonal nonzero in column $j$ of $L$. If column $j$ has no off-diagonal nonzero, we set $\gamma[j] = j$. (Hence $\gamma[n] = n$.)

We now define an *elimination tree* corresponding to the structure of $L$. The tree has $n$ nodes, labelled from 1 to $n$. For each $j$, if $\gamma[j] > j$, then node $\gamma[j]$ is the *parent* of node $j$ in the elimination tree, and node $j$ is one of possibly several *child* nodes of node $\gamma[j]$. We assume that the matrix $A$ is *irreducible* , so that $n$ is the only node with $\gamma[j] = j$ and it is the *root* of the tree. Thus, for $1 \le j < n$, $\gamma[j] > j$.

The importance of the elimination tree is that it gives precise information about the column dependencies. In particular, the computation of column $i$ cannot be completed until the calculation of all columns corresponding to descendent nodes of node $i$ have been completed. On the other hand, columns corresponding to nodes at the same level of the tree are *independent*, and can be computed in parallel.

# Elimination trees - an example

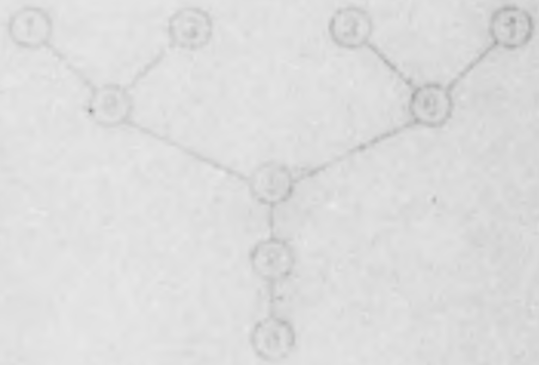


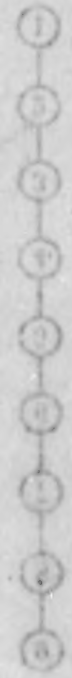
Two orderings and their corresponding factors.

$$\begin{pmatrix}
\times &&&&&&&& \\
& \times &&&&&&& \\
&& \times &&&&&& \\
&&& \times &&&&& \\
\times & \times &&& \times &&&& \\
&& \times & \times && \times &&& \\
\times && \times && \times & \times & \times && \\
\times & \times & \times & \times & \times & \times & \times & \times & \\
& \times && \times & \times & \times & \times & \times & \times
\end{pmatrix}
\begin{pmatrix}
\times &&&&&&&& \\
\times & \times &&&&&&& \\
& \times & \times &&&&&& \\
\times & \times & \times & \times &&&&& \\
\times & \times & \times & \times & \times &&&& \\
& \times & \times & \times & \times & \times &&& \\
&&& \times & \times & \times & \times && \\
&&& \times & \times & \times & \times & \times & \\
&&&& \times & \times & \times & \times & \times
\end{pmatrix}$$

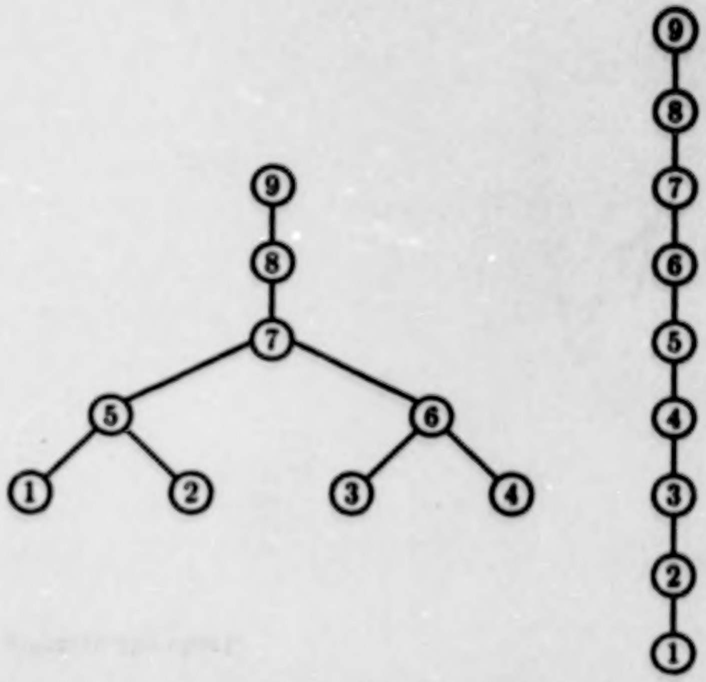
# Elimination trees - an example

In order to see the role that elimination trees might play in identifying parallelism, consider two different orderings of a 3 by 3 grid problem. The 9 vertices of the grid are numbered in some manner, and the associated matrix $A$ has the property that $a_{ij} \neq 0$ if and only if vertex $i$ and vertex $j$ are associated with the same small square in the grid. Two different orderings of the grid along with their associated Cholesky factors are given in the chart.





# Elimination trees - an example


9
8
7
5
6
1
2
3
4
9
8
7
6
5
4
3
2
1

The elimination trees associated with the matrices.

# Elimination trees - an example

**The elimination tree on the left is typical of those generated by orderings that are good in the sense of yielding low fill and low operation counts. Its tree structure is short and wide, and such trees and their associated orderings lend themselves well to parallel computation. For example, it should be clear that columns 1,2,3 and 4 can be computed in parallel. Moreover, when they have been computed, columns 5 and 6 can be computed in parallel.**

**On the other hand, the band-oriented ordering shown above is undesirable because it imposes the same serial execution that is imposed in the dense case. Moreover, the operation counts and fill-in are inferior to that of the first ordering.**

**In the elimination tree, if node $i$ and node $j$ belong to the same level of the tree, it is clear that the computation of columns $i$ and $j$ can be performed independently so long as the tasks associated with their descendant nodes have all been completed. In order to gain high processor utilization, it is therefore desirable to assign, if possible, nodes on the same level of the tree to different processors.**

# Elimination trees ... facts

- *All* labellings of the elimination tree that number child nodes before their parents are *equivalent*. They produce the same fill and the same arithmetic operation count.

- Elimination trees can be computed *very* rapidly in serial mode.

- Good orderings may not yield balanced trees.

- Elimination trees can be rearranged (restructured) to enhance the available parallelism ⋯ important recent work by Joseph Liu, who has shown how to restructure trees to change their height but preserve the fill and operation counts.

# Elimination trees ... facts

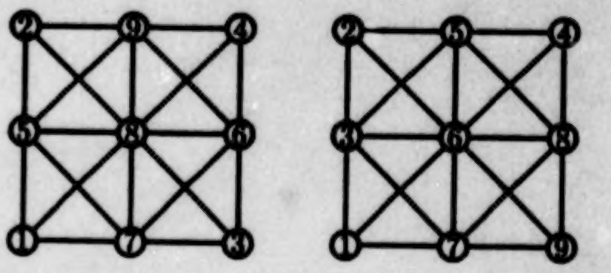
It is generally known (although there seems to be no published proof) that given an ordering of a sparse matrix, and therefore an elimination tree, any symmetric *reordering* of the matrix based on a relabelling of the tree that numbers each vertex ahead of its parent is *equivalent* to the original ordering in terms of fill and computation.

In some contexts it is necessary to be able to obtain the elimination tree prior to determining the structure of $L$. Fortunately, there is a very efficient algorithm that can be used to obtain the tree directly from the structure of the matrix $A$ due to Liu. Its complexity can be shown to be $O(|A|\log_2 n)$.

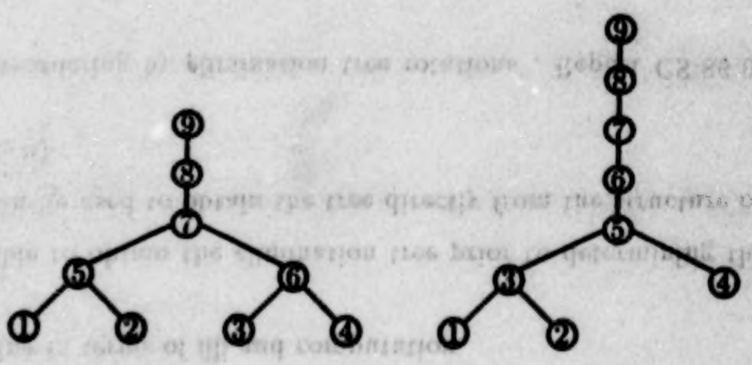
J.W-H. Liu, "Equivalent sparse matrix reordering by elimination tree rotations", Report CS-86-09, Dept of Computer Science, York University.

J.W-H. Liu, "Reordering sparse matrices for parallel elimination", Report CS-87-02, Dept of Computer Science, York University.

# Elimination trees - restructuring


2
9
4
5
8
6
1
7
3
2
5
4
3
6
8
1
7
9

Two *equivalent* orderings and their corresponding trees.


9
8
7
5
6
1
2
3
4
9
8
7
6
5
3
4
1
2

# Elimination trees ... restructuring

Recently Liu has shown how to produce equivalent reorderings of matrices which *change* the elimination tree. That is, the amount of fill and the amount of computation required for the factorization do not change, but the structure of the tree does change. He has also developed a fast algorithm which will reorder a sparse matrix problem in order to either increase or decrease the height of the elimination tree, while preserving the level of fill and computation.

The chart shows two orderings of the 3 × 3 grid problem which produce exactly the same fill, but which have different elimination trees. The ordering on the left obviously lends itself to parallel computation somewhat better than the one on the right. In other contexts, such as when auxiliary storage or virtual memory is used, "tall, skinny" trees are desirable because this tends to reduce the amount of main storage that is required. Liu's restructuring algorithm is useful in this situation as well.

J. W-H. Liu, "A note on sparse factorization in a paging environment", Report CS-86-13, Dept. of Computer Science, York University.

# An overall strategy

1. **Find a good sparsity-preserving ordering for *A*.**
2. **Find the elimination tree corresponding to the reordered *A*.**
3. **Reorder the problem so as to reduce the height of its elimination tree using Liu's height-reducing algorithm.**
4. **Assign the column tasks to the processors in a "bottom up" manner with respect to the tree, but as much as possible, assign subtrees of the tree to subsets of the processors.**

# An overall strategy

**In view of the previous observations, it would seem desirable to first find the best ordering in terms of fill and computation, and then within the class of reorderings that preserve that level of fill and computation, choose one which produces a "short" and "wide" tree.**

## Solving Indefinite Sparse Systems

- Problem to be solved: $Ax = b$.
- If $A$ has no special properties, some form of pivoting is required.
- Important fact: $A$ normally suffers *fill* during the factorization.
- Important fact: permuting the rows and columns of $A$ can have an enormous effect on the amount of fill that occurs.
- Standard solution: Compute a factorization of $P_rA$ or $P_rAP_c$, where $P_r$ and $P_c$ are $n \times n$ permutation matrices. Permutations are determined *during the factorization* by a combination of numerical stability and sparsity requirements.
- Some form of *threshhold pivoting* is usually employed.

# Solving Indefinite Sparse Systems

When Gaussian elimination is applied to a sparse matrix $A$, it normally suffers *fill*. That is, its factors usually have nonzeros in positions which are zero in the corresponding positions in $A$.

If $A$ has no special properties, it is well-known that applying Gaussian elimination to $A$ may fail. A zero element might turn up in the pivot position. In this case, some form of row and/or column pivoting must be performed in order to ensure numerical stability..

Permuting the rows and columns of $A$ can have an enormous effect on the amount of fill that occurs during the factorization. Note that here and at all times we make the usual *no-cancellation assumption*. That is, whenever two nonzero quantities are added or subtracted, the result is nonzero. This means that in the analysis we ignore any zeros which might be created through exact cancellation. Such cancellation rarely occurs, and any such prediction would be difficult in general, particularly in floating point arithmetic which is subject to rounding errors.

Given $A$, one normally obtains a factorization of $P_rA$ or $P_rAP_c$, where $P_r$ and $P_c$ are $n \times n$ permutation matrices. When $A$ is sparse, these permutations are determined *during the factorization* by a combination of (usually competing) numerical stability and sparsity requirements. Dynamic data structures obviously are required. Different matrices, even though they may have the same nonzero pattern, will normally yield different $P_r$ and $P_c$, and therefore have factors with different sparsity patterns.

The use of dynamic data structures tends to lead to very complicated code, and substantial computational and storage overhead.

## Solving Indefinite Systems - Partial Pivoting

- Gaussian elimination with partial pivoting yields:

$$A = P_1 M_1 P_2 M_2 \cdots P_{n-1} M_{n-1} U,$$

where $P_k$ is an elementary permutation matrix corresponding to the row interchange performed at step $k$, $M_k$ is a unit lower triangular matrix whose $k$-th column contains the multipliers used at the $k^{th}$ step, and $U$ is an upper triangular matrix.

- Structures of $M = \Sigma_{k=1}^{n-1} M_k$ and $U$ depend on the interchanges, which in turn depends on the numerical values of $A$.

# Solving Indefinite Systems - Partial Pivoting

A standard approach for solving $Ax = b$ involves reducing $A$ to upper triangular form using elementary row eliminations (i.e., Gaussian elimination). In order to maintain numerical stability, one may have to interchange rows at each step of the elimination process. Thus, we may express the result of the process as follows:

$$A = P_1 M_1 P_2 M_2 \cdots P_{n-1} M_{n-1} U,$$

where $P_k$ is an $n \times n$ elementary permutation matrix corresponding to the row interchange performed at step $k$, $M_k$ is an $n \times n$ unit lower triangular matrix whose $k$-th column contains the multipliers used at the $k^{th}$ step, and $U$ is an $n \times n$ upper triangular matrix. As noted earlier, when $A$ is sparse, fill normally occurs during the triangular decomposition, so there are usually collectively more nonzeros in $M = \sum_{k=1}^{n-1} M_k$ and $U$ than in $A$. Moreover, the structures of $M = \sum_{k=1}^{n-1} M_k$ and $U$ depend on the interchanges, which in turn depend on the numerical values of $A$.

## Solving Indefinite Systems - an alternative approach

- Create from the ***structure*** of $A$ a data structure which can accommodate all the nonzeros in $M$ and $U$, ***irrespective of the actual pivot sequence chosen.*** Denote matrices with these structures by $\bar{L}$ and $\bar{U}$.

- Carry out the factorization of $A$ using this static data structure.

- Experience (and some theoretical results) show that the amount by which the data structure is "too big" is not excessive.

- A static data structure is efficient in terms of both space and computation.

- This approach has been implemented and extensively tested.

- One can define an elimination tree for the matrix $\bar{U}$ that can be used to guide the exploitation of parallelism.

## Solving Indefinite Systems - an alternative approach

The basic strategy is to create from the *structure* of $A$ a data structure which can accommodate all the nonzeros in $M$ and $U$, *irrespective of the actual pivot sequence chosen.* Let $\bar{L}$ and $\bar{U}$ be matrices whose structures contain respectively the structures of $M$ and $U$, irrespective of the pivot sequence $P_1, P_2, \cdots, P_{n-1}$. Although the data structure for these matrices is more generous than it needs to be for any *specific* sequence, it can be set up in advance of the numerical computation, which can then be done efficiently using a static data structure. The advantages noted earlier for positive definite matrices are then be available.

This approach has been implemented and tested, and shown to be very competitive with alternative approaches for solving indefinite finite element problems. Work is currently under way to implement the scheme on a shared memory multiprocessor. One can define an elimination tree for the matrix $\bar{U}$ that can be used to guide the exploitation of parallelism in much the same manner as discussed earlier in the context of solving symmetric positive definite systems on parallel architectures.

# Selected Publications

1. Alan George, Michael T. Heath and Joseph W-H. Liu, "Parallel Cholesky Factorization on a Shared-Memory Multiprocessor", Linear Algebra and its Applics., 77 (1986), pp. 165-188.

2. Alan George and Esmond Ng, "Symbolic Factorization for Sparse Gaussian Elimination with Partial Pivoting", SIAM J. Sci. and Stat. Computing, 8 (1987), pp. 877-898.

3. Alan George, Michael T. Heath, Joseph W-H. Liu and Esmond Ng, "Symbolic Cholesky Factorization on a Local-Memory Multiprocessor", Parallel Computing, 5 (1987), pp. 85-95.

4. Eleanor Chu and Alan George, "Gaussian Elimination with Partial Pivoting and Load Balancing on a Multiprocessor", Parallel Computing, 5 (1987), pp. 65-74.

5. Alan George, Michael T. Heath, Joseph Liu and Esmond Ng, "Sparse Cholesky Factorization on a Local-Memory Multiprocessor", SIAM J. Sci. and Stat. Computing, (to appear).

6. Alan George and Joseph Liu, "The Evolution of the Minimum Degree Ordering Algorithm", SIAM Review, (to appear).

7. Alan George and Esmond Ng, "On the Complexity of Sparse QR and LU Factorization of Finite Element Matrices", SIAM J. Sci. and Stat. Computing, (to appear).

8. Alan George, Joseph Liu and Esmond Ng, "Communication Results for Parallel Sparse Cholesky Factorization on a Hypercube", Parallel Computing, (submitted).

N89
29782
UNCLAS

DD652564 N89-29782

54-39

2 P.

211950

# PARALLEL EIGENVALUE EXTRACTION

Fred A. Akl
Ohio University

## Abstract

~~This report presents~~ a new numerical algorithm for the solution of large-order eigenproblems typically encountered in linear elastic finite element systems is presented. The architecture of parallel processing is utilised in the algorithm to achieve increased speed and efficiency of calculations. The algorithm is based on the frontal technique for the solution of linear simultaneous equations and the modified subspace eigenanalysis method for the solution of the eigenproblem. Assembly, elimination and back-substitution of degrees of freedom are performed concurrently, using a number of fronts. All fronts converge to and diverge from a predefined global front during elimination and back-substitution, respectively. In the meantime, reduction of the stiffness and mass matrices required by the modified subspace method can be completed during the convergence/divergance cycle and an estimate of the required eigenpairs obtained. Successive cycles of convergence and divergence are repeated until the desired accuracy of calculations is achieved. The advantages of this new algorithm in parallel computer architecture are discussed.

## Generalized Eigenproblem

$$[K][\Phi] = [M][\Phi][\Omega]$$

N - degrees of freedom

Required n eigenpairs, $n \leq N$

[K] positive-definite

- New parallel algorithm for the solution of large scale eigenproblems in finite element applications.
- Work is in progress to implement algorithm on NAS Cray 2 computer at Ames.

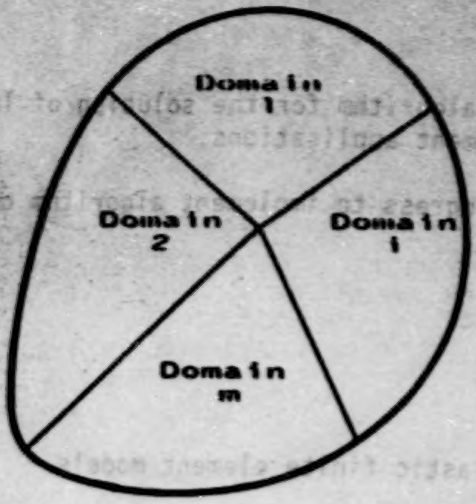
- **Assumptions**

  1 - Linear elastic finite element models

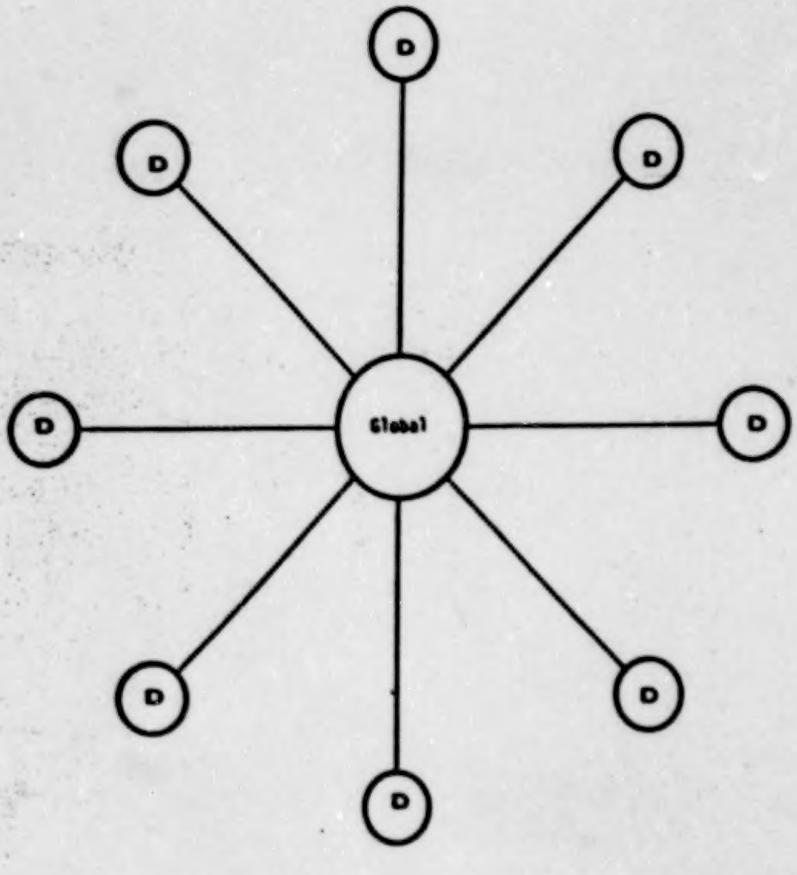
  2 - n lower order eigenpairs are required, i.e. $\omega_1^2 \leq \omega_2^2 \leq \ldots \omega_n^2$

  3 - [K] is positive-definite

  4 - [M] is semi-positive definite


Domain
1
Domain
2
Domain
i
Domain
m

Finite Element Model Subdivided into m Domains


D
D
D
D
Global
D
D
D
D
D

Domain i

- **Consider a parallel computer with (m+1) processors (tasks).**
- **Designate the first processor as a global processor (task).**
- **Designate the remaining m-processors as domain processors (tasks).**
- **A finite element model can be divided into a number of domains equal to m.**
- **A star architecture (or tree) is the first to be investigated.**

$$[K][\Phi] = [M][\Phi][\Omega]$$

1 - Creation of $K^e$ & $M^e$

2 - Eigensolution (Modified Subspace)

3 - Equation Solver (Frontal Solution)

- **Three major steps of large computational requirements:**

  **1 - Creation of element stiffness and mass matrices.**

  **2 - Extraction of a set of eigenpairs.**

  **3 - Solution of a set simultaneous linear equations.**

- **The merits of selecting the modified subspace method for step #2 and the frontal solution for step #3 above all discussed in the next new graphs.**

## Modified Subspace Method

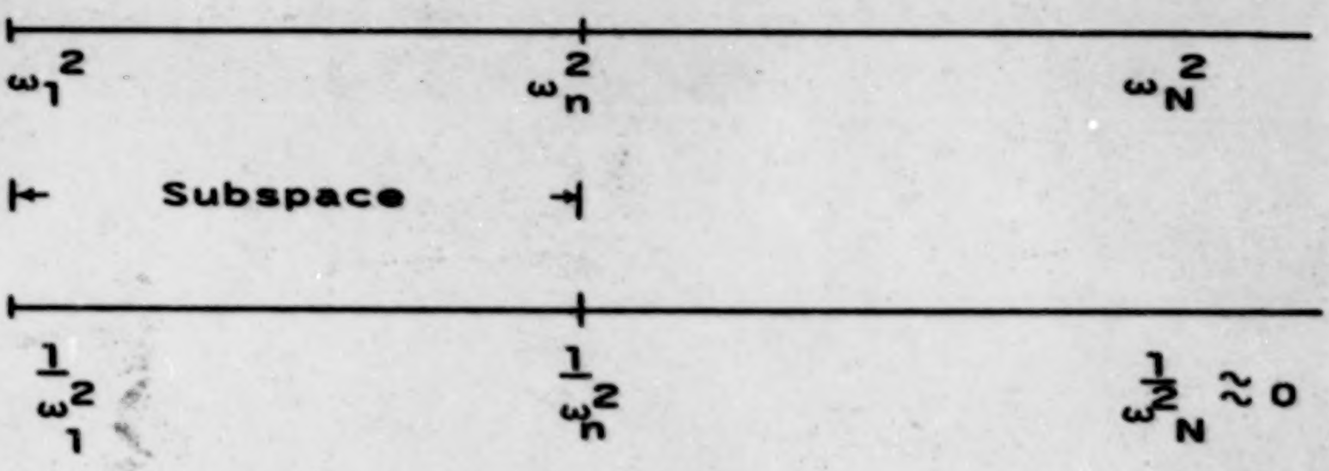
$$[V]^*_{\ell+1} = ([K]^{-1}[M] - \beta_\ell[I])[V]_\ell$$

$$= [K]^{-1}[B]_\ell - \beta_\ell[V]_\ell$$

where $\ell = 1, 2, 3, \ldots\ldots$

$$\beta_\ell = 1/2(1+r_{\ell-1})/\omega_n^2$$

$$\omega_1^2 \le \cdots\cdots \le \omega_n^2 \le \omega_{n+1}^2 \le \cdots\cdots \le \omega_N^2$$


$\omega_1^2$
$\omega_n^2$
$\omega_N^2$
Subspace
$\frac{1}{\omega_1^2}$
$\frac{1}{\omega_n^2}$
$\frac{1}{\omega_N^2} \approx 0$

The Modified Subspace method iterates simultaneously for a subset of eigenpairs $[\phi,\omega]$ of the generalized eigenproblem:

1 - Let $[V]_1$ be a starting eigenvectors. Experience has shown that random numbers can be used here. A number of techniques are available in literature for selecting $[V]_1$.

2 - Operate on each $[V]_\ell$ as follows

$$[V]^*_{\ell+1} = [K]^{-1}[M][V]_\ell = [K]^{-1}[B]_\ell$$

where $\ell = 1, 2, 3, \ldots$

3 - Modify $[V]^*_{\ell+1}$ to increase convergence rate by one third on average

$$[V]^*_{\ell+1} \leftarrow V^*_{\ell+1} - \beta_\ell V_\ell$$

where: $\beta_\ell = 0$ for $\ell=1$ and $\ell>11$

$\beta_\ell = 0.5\ (1+r_{\ell-1})/\omega_n^2 \quad 1\leq\ell\leq 11$

$r_{\ell-1}$ are the interval points of the 11-th order Labatto rule $[-1, 1]$

Roots of the 11th Order Lobatto Rule (Kopal 1961)

| | | | |
|---|---|---|---|
| $r_1$ | -0.9533098466 | $r_6$ | 0.0000000000 |
| $r_2$ | -0.8463475646 | $r_7$ | +0.2492869301 |
| $r_3$ | -0.6861884690 | $r_8$ | +0.4829098210 |
| $r_4$ | -0.4829298210 | $r_9$ | +0.6861884690 |
| $r_5$ | -0.2492869301 | $r_{10}$ | +0.8463475646 |

## Subspace

$$[K]^{*}_{\ell+1} = \Sigma[V]^{eT}_{\ell+1}[K]^{e}[V]^{e}_{\ell+1}$$

$$[M]^{*}_{\ell+1} = \Sigma[V]^{eT}_{\ell+1}[M]^{e}[V]^{e}_{\ell+1}$$

## The Auxiliary Eigenproblem

$$[K]^{*}_{\ell+1}\ [Q]_{\ell+1} = [M]^{*}_{\ell+1}\ [Q]_{\ell+1}\ [\Omega]_{\ell+1}$$

## Improved Eigenvectors

$$[V]^{e}_{\ell+1} = [V]^{*e}_{\ell+1}\ [Q]_{\ell+1}$$

4 - Project K and M onto the required subspace.

5 - Solve the auxiliary eigenproblem to obtain $[Q]_{\ell+1}$ and $[\Omega]_{\ell+1}$.

6 - An improved set of eigenvectors $[V]_{\ell+1}$ can be obtained.

7 - Test for convergence on $\omega_n^2$. Repeat steps 2 to 6 until desired accuracy is achieved.

**Note**

1. Step #2 is performed using the frontal solution, concurrently within each domain.

2. Steps 1, 3, 4 and 6 are processed concurrently within each domain.

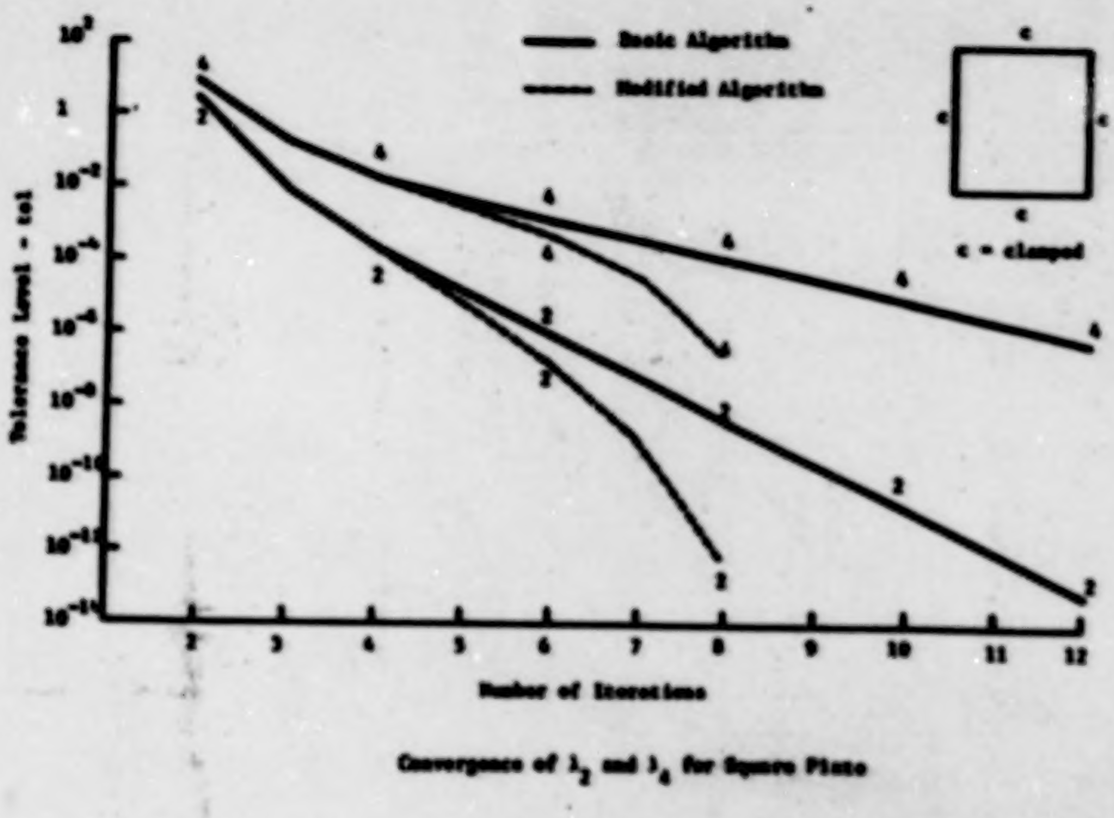
# RATE OF CONVERGENCE

- **Basic Subspace**

$$\leq \frac{\omega_1^2}{\omega_{n+1}^2}$$

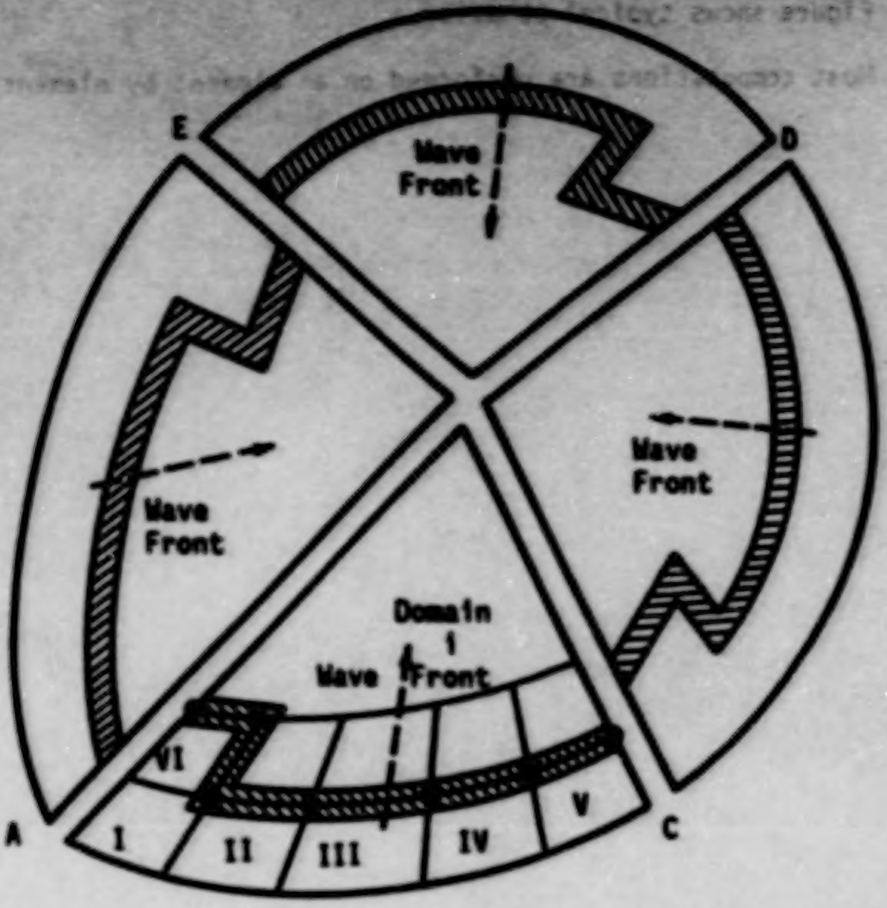
- **Modified Subspace**

$$\leq \frac{\omega_1^2}{\omega_{n+1}^2} \; \frac{1-\beta_\ell \omega_{n+1}^2}{1-\beta_\ell \omega_1^2}$$



Convergence of $\lambda_2$ and $\lambda_4$ for Square Plate

- **Rate of convergence of the modified subspace is 33% faster on average compared to the classical subspace method.**
- **Figure shows typical behavior.**
- **Most computations are performed on an element by element basis.**


E
D
Wave
Front
Wave
Front
Wave
Front
Wave
Front
Domain
1
Wave
Front
A
C
VI
I
II
III
IV
V

Multi-Frontal Parallel Processing

## Frontal Solution

1 - Gauss elimination technique.

2 - Underlying philosophy is based on processing of elements one by one.

3 - Simultaneous assembly and elimination of variables.

4 - The optimum frontal width is at most equal to the optimum band width.

5 - Numbering of nodes has no impact on optimality while numbering of elements is important to minimize the frontal width.

6 - More efficient for solid elements and elements with mid-side nodes.

7 - It requires a pre-front to determine last appearance of each node.

8 - It lends itself to parallel solutions.

# Multi-Frontal Solution

- **Within each domain**

$$\hat{k}^{d}_{ij} \leftarrow k^{d}_{ij} - \Sigma \left[ \frac{k_{is}k_{sj}}{k_{ss}} \right]^{d}$$

$$\hat{b}^{d}_{iq} \leftarrow b^{d}_{iq} - \Sigma \left[ \frac{k_{is}b_{sq}}{k_{ss}} \right]^{d}$$

**For domain i**

$$[K][V]^*_{\ell+1} = [B]_{\ell+1}$$

Assembly and elimination gives

$$U_i V_d^* + \hat{K}_{dF} V_F^* = \hat{B}_d$$

$$\hat{K}_{FF} V_F^* = \hat{B}_F$$

where $U_i$ upper Δ matrix for domain i

$V_d^*$ variables within domain i

$V_F^*$ variables along global front of domain i

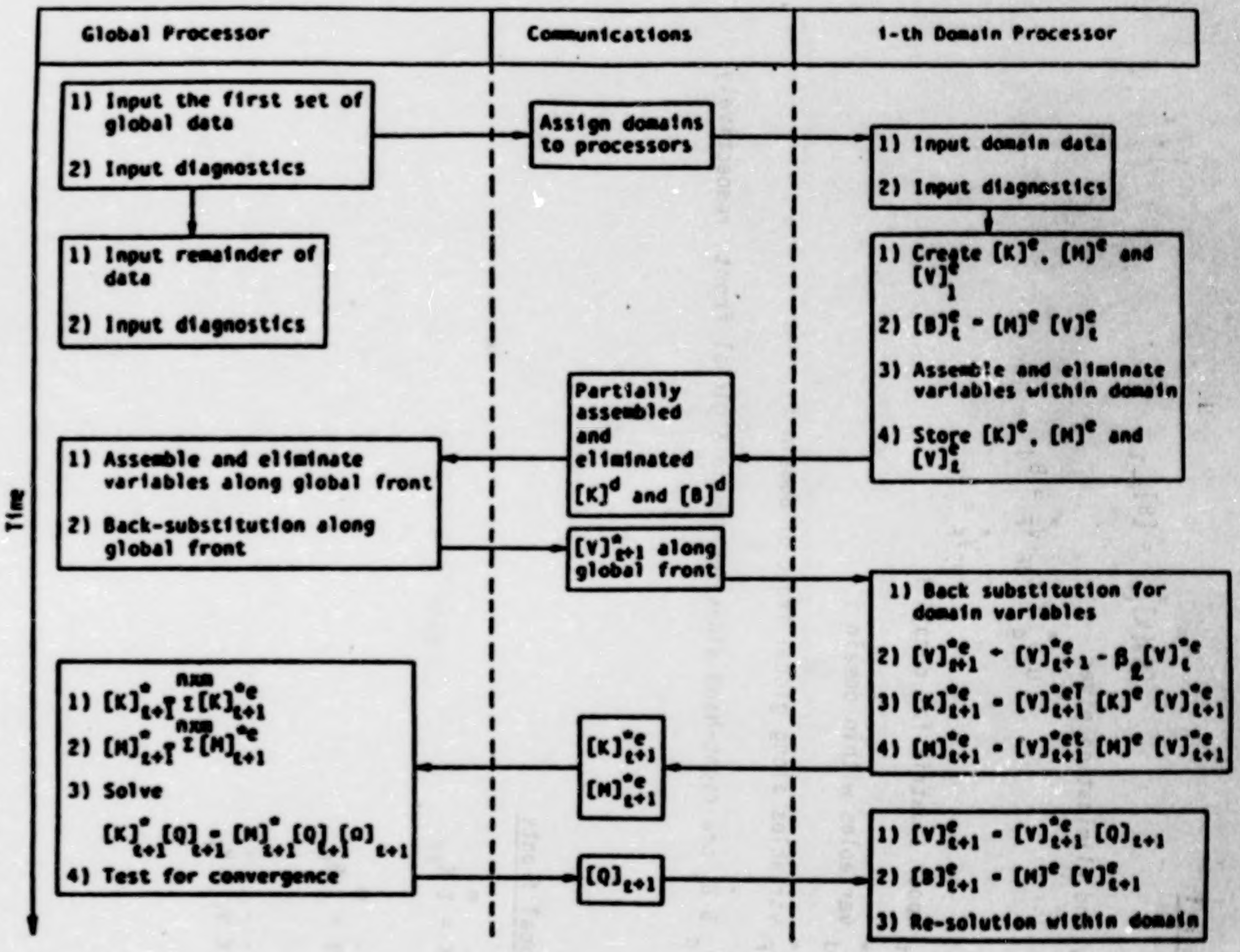
$\hat{B}_d$ & $\hat{B}_F$ are right-hand sides for domain & global front, respectively

**For global fronts**

$$\hat{K} = \sum^{m} K_{FF}$$

$$\hat{B} = \sum^{m} B_F$$

$$\hat{K}\,\hat{V}_F = \hat{B}_F$$


Global Processor
Communications
i-th Domain Processor
1) Input the first set of global data
2) Input diagnostics
Assign domains to processors
1) Input domain data
2) Input diagnostics
1) Input remainder of data
2) Input diagnostics
1) Create $[K]^e$, $[M]^e$ and $[V]_1^e$
2) $[B]_\ell^e = [M]^e [V]_\ell^e$
3) Assemble and eliminate variables within domain
4) Store $[K]^e$, $[M]^e$ and $[V]_\ell^e$
Partially assembled and eliminated $[K]^d$ and $[B]^d$
1) Assemble and eliminate variables along global front
2) Back-substitution along global front
$[V]_{\ell+1}^e$ along global front
1) Back substitution for domain variables
2) $[V]_{\ell+1}^{*e} = [V]_{\ell+1}^{*e} - \beta_\ell [V]_\ell^{*e}$
3) $[K]_{\ell+1}^{*e} = [V]_{\ell+1}^{*eT} [K]^e [V]_{\ell+1}^{*e}$
4) $[M]_{\ell+1}^{*e} = [V]_{\ell+1}^{*eT} [M]^e [V]_{\ell+1}^{*e}$
$[K]_{\ell+1}^{*e}$
$[M]_{\ell+1}^{*e}$
1) $[K]_{\ell+1}^* = \sum^{nxm} [K]_{\ell+1}^{*e}$
2) $[M]_{\ell+1}^* = \sum^{nxm} [M]_{\ell+1}^{*e}$
3) Solve
$[K]_{\ell+1}^* [Q]_{\ell+1} = [M]_{\ell+1}^* [Q]_{\ell+1} [\Omega]_{\ell+1}$
4) Test for convergence
$[Q]_{\ell+1}$
1) $[V]_{\ell+1}^e = [V]_{\ell+1}^{*e} [Q]_{\ell+1}$
2) $[B]_{\ell+1}^e = [M]^e [V]_{\ell+1}^e$
3) Re-solution within domain
Time

Computational Tasks and Communications

- Successful implementation of the new parallel algorithm depends on:

  1 - Maximizing the efficiency of communication links between the global task and the domains

  2 - Minimizing sequential computational steps

  3 - Multi-threaded I/O

- Final report will be available in the Summer 1988

## Anticipated Benefits

- Parallel eigenvalue extraction algorithm to maximize efficiency and speed-up of computations.
- A general purpose eigenproblem solver for finite element analysis in parallel computing environment.

op. sys.
start

copy data file | copy data file

| **GLBFRONT** | **DOMFRONT** |
|---|---|
| 1. read/check first data card | 1. read/check first data card |
| 2. set-up VEC for data input | 2. set-up VEC for data input |
| 3. data input and check | 3. data input and synthesis |
| 4. reset VEC for global fronts | 4. reset VEC for domain |
| 5. pre-front for global fronts | 5. pre-front for domain |
| | 5. element K and M matrices |
| | 6. domain assembly/elimination |
| | 7. $K_{FF}$ to GLBFRONT |
| 6. global fronts solution | |
| 7. $V_F$ to DOMFRONT | |
| | 8. domain solution and subspace |
| | 9. $K_d^*$ and $M_d^*$ to DOMFRONT |
| 8. subspace solution | |
| 9. Q to DOMFRONT | |
| 10. convergence test | 10. Improved eigenvectors $V^e$ |

N89
29783
UNCLAS

N89-29783

510-39

128

211951

D4859255

# Parallel Algorithms and Architectures
# for
# Computational Structural Mechanics

(Grant. No. NAG-1-466)

Merrell Patrick, Principal Investigator
Shing MA, Graduate Research Assistant
Umesh Mahajan, Graduate Research Assistant

## Abstract

The determination of the fundamental (lowest) natural vibration frequencies and associated mode shapes is a key step used to uncover and correct potential failures or problem areas in most complex structures. However, the computation time taken by finite element codes to evaluate these natural frequencies is significant, often the most computationally intensive part of structural analysis calculations. There is continuing need to reduce this computation time. This study addresses this need by developing methods for parallel computation.

PRECEDING PAGE BLANK NOT FILMED

PAGE 260 INTENTIONALLY BLANK

# DUKE'S CSM SUPPORTED ACTIVITY

- **SPECIFIC OBJECTIVES**
- **METHODS**

# DUKE'S CSM SUPPORTED ACTIVITY

Duke's research group is developing parallel methods for solving the generalized eigenproblem of the form $Kx = \lambda Mx$, where $K$ and $M$ are symmetric and $M$ is positive definite. Two methods are being implemented. They are based on subspace iteration (see ref. 2) and spectrum sectioning (see ref. 3). The methods are being tested with stiffnesses matrices, $K$, and mass matrices, $M$, obtained from NICE/SPAR runs on two focus problems, space mast problem and stiffened panel problem. The research is closely coordinated with that of the NASA Langley CSM group (see ref. 1).

ORIGINAL PAGE IS
OF POOR QUALITY

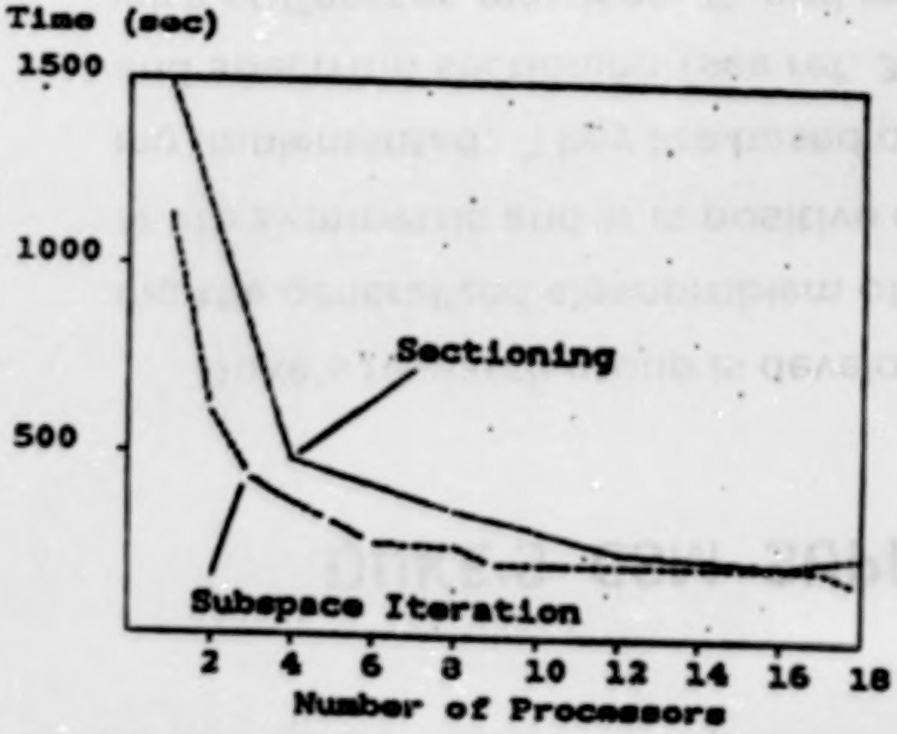
# EXPERIMENTAL RESULTS - I

**SOLVE : $Kx = \lambda Mx$ (10 lowest eigenpairs)**

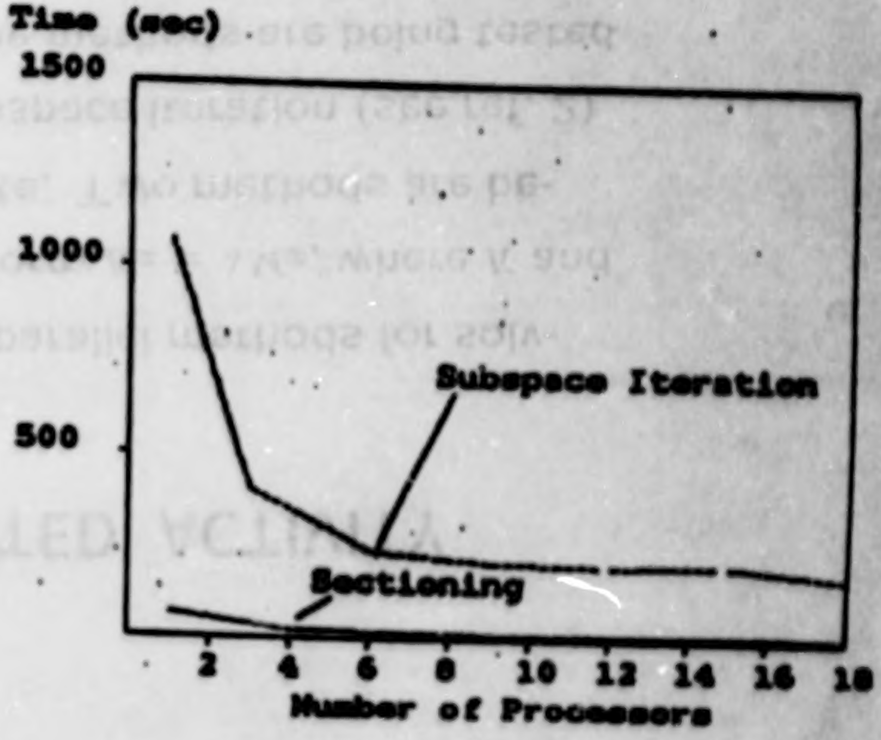
**PARALLEL METHODS : Sectioning**
**: Subspace Iteration**

**Execution Time Vs Number of Processors**

**54-Bay Space Mast**
**K : dimension 486, bandwidth 35**
**M : diagonal**



**One Dimensional Poisson's Equation**
**K : dimension 486, bandwidth 3**
**M : I**



# EXPERIMENTAL RESULTS – I

The graphs pictured above compare execution times versus number of processors of the parallel sectioning and subspace iteration methods for two different sets of test matrices. Objective of the commparison is to show the impact of the bandwidth on performance of the methods.

Clearly, bandwidth affects the relative performance of the two methods with parallel sectioning having superior performance for the small bandwidth problem but losing that superiority to the parallel subspace iteration method for the large bandwidth problem.

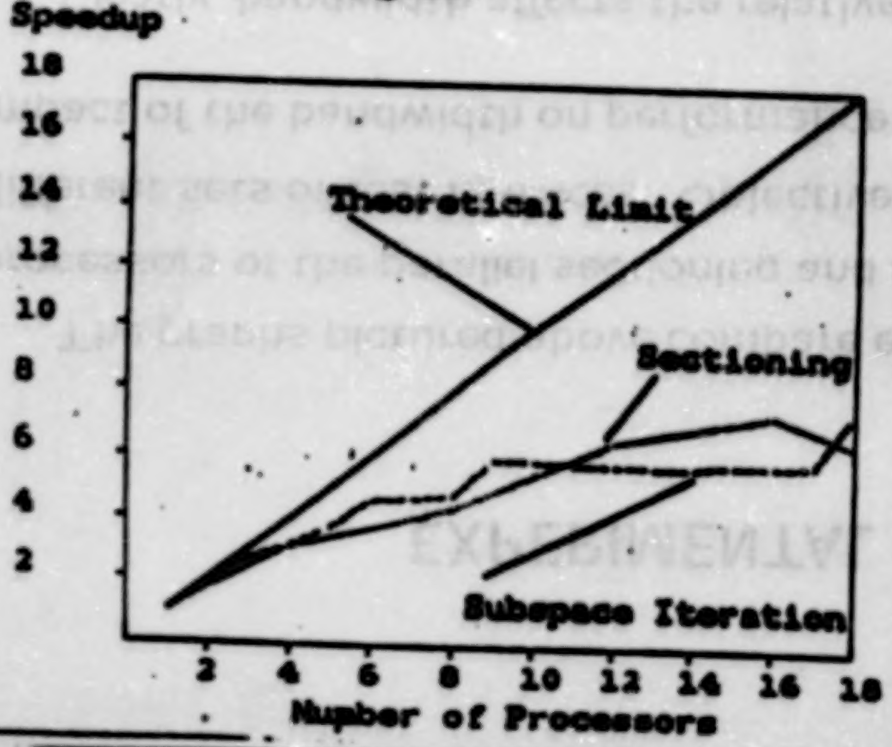
## EXPERIMENTAL RESULTS - II

**SOLVE : $Kx = \lambda Mx$ (10 lowest eigenpairs)**

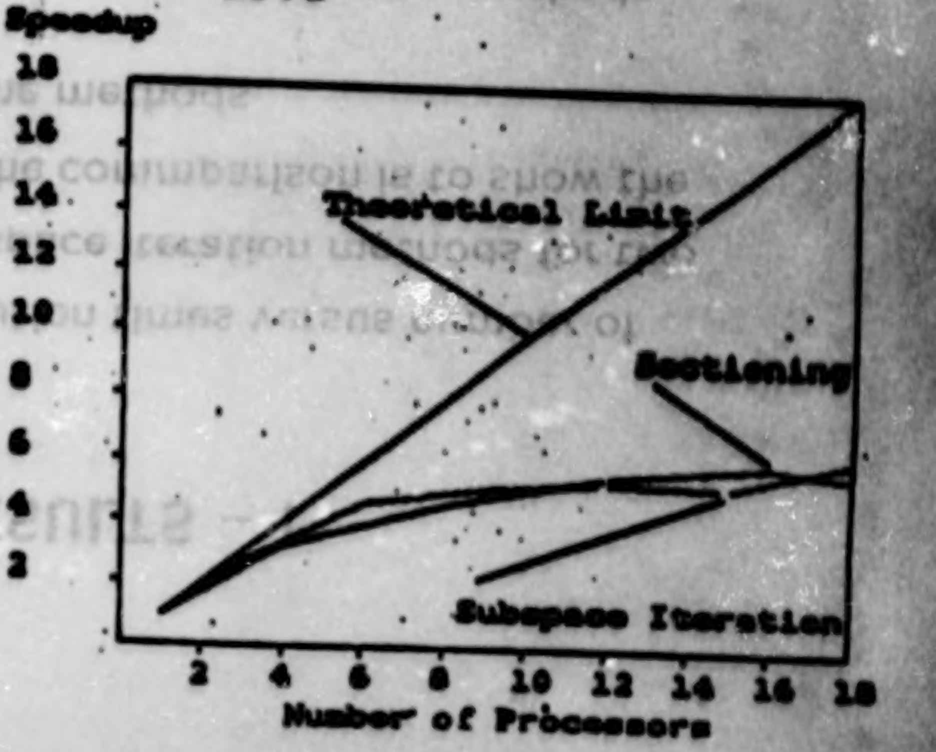
**PARALLEL METHODS : Sectioning**
**: Subspace Iteration**

**Speedup Vs Number of Processors**

**54-Bay Space Mast**
**K : dimension 486, bandwidth 35**
**M : diagonal**



**One Dimensional Poisson's Equation**
**K : dimension 486, bandwidth 3**
**M : I**



# EXPERIMENTAL RESULTS – II

Speedup versus number of processors of the parallel sectioning and subspace iteration methods for two sets of matrices are compared. It should be noted that the parallel sectioning method cannot yield a speedup higher than the number of eigenvalues being computed, 10 in this case. Hence parallel sectioning method comes closer to its theoretical maximum than does the subspace iteration method.

# EXPERIMENTAL RESULTS - III

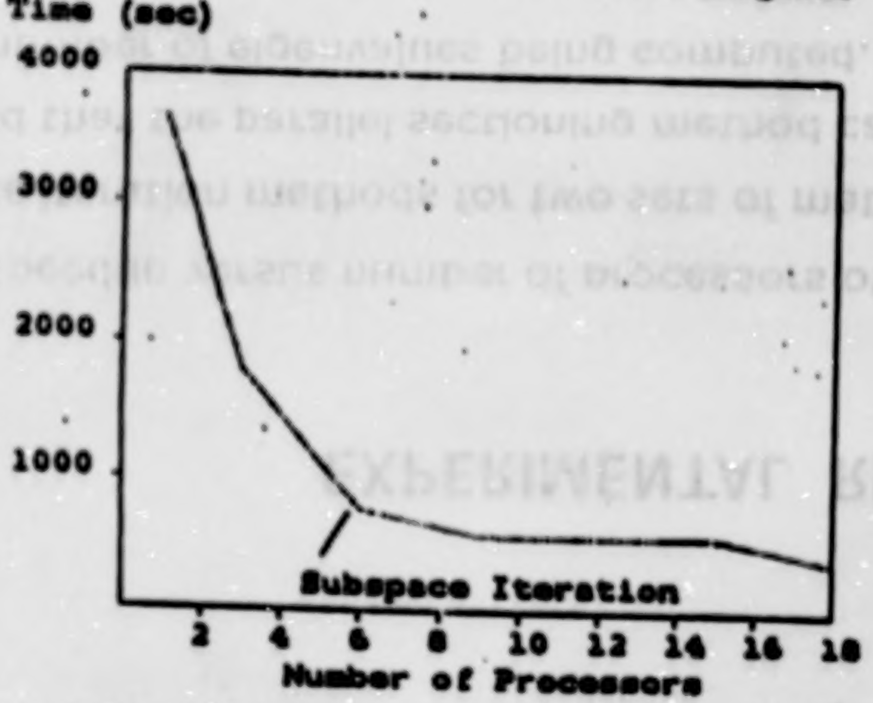
**SOLVE : $Kx = \lambda Mx$ (10 lowest eigenpairs)**

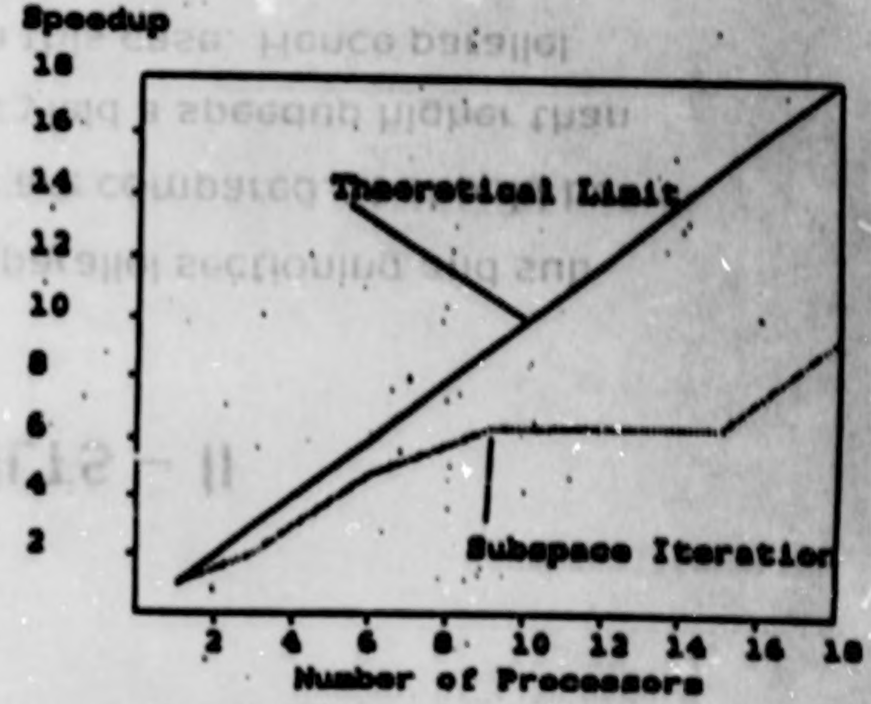
**PARALLEL METHOD : Subspace Iteration**

**Execution Time Vs Number of Processors**

**Speedup Vs Number of Processors**

**Stiffened Epoxy Panel**
**K : dimension 456, bandwidth 228**
**M : diagonal**





# EXPERIMENTAL RESULTS – III

Both execution time and speedup versus number of processors of the parallel subspace iteration method for the stiffened epoxy panel problem with 456 degrees of freedom and bandwidth are shown. Corresponding graphs for the parallel sectioning were not included because its execution time for this higher bandwidth problem had grown to hours, which indicates its unsuitability for solving high bandwidth problems. The increased execution time is due to the increased time required to factor $K - MM$ for higher bandwidth $K$ and $M$. A factorization is required each time the spectrum is sectioned or sliced (see ref. 1).

# CURRENT AND FUTURE EFFORTS

- LARGE SCALE MODELS OF FOCUS PROBLEMS
- IMPROVEMENT OF METHODS
- COMPARISON OF PARALLEL LANGUAGES FOR SCIENTIFIC COMPUTING

# CURRENT AND FUTURE EFFORTS

- **Large Scale Models of Focus Problems**

  Our parallel methods for solving the generalized eigenvalue problem are being tested on larger scale models of the two focus problems. With the FLEX/32 upgrade, much larger problems can now be solved. How much larger is yet to be determined.

- **Improvement of Methods**

  The parallel subspace iteration method is being improved by adding shifting to the inverse iteration loop of the method. This was not added initially because shifting requires solutions of indefinite linear systems. This change should decrease substantially the number of iterations and improve the overall performance of subspace iteration.

- **Comparison of Parallel Languages for Scientific Computing**

  In addition to the development of parallel methods, we are comparing different parallel programming languages available. These languages include Concurrent FORTRAN, Force, PISCES, and Schedule. Points of comparison include expression of functional and data parallelism, communication annd synchronization mechanisms, ease of learning language, readability of program, and testing and debugging support.

# REFERENCES

1. Knight, N.F., and Stroud, W.J., "Computational Structural Mechanics: A New Activity at the NASA Langley Research Center," NASA TM 87612, September, 1985.

2. Bathe, K., Finite Element Procedures in Engineering Analysis, Prentice-Hall, Inc., 1982.

3. Peters, G., and Wilkinson, J.H., "Eigenvalues of $Ax = \lambda Bx$ with Band Symmetric $A$ and $B$," Computing Journal, Vol. 12, 1969, pp. 388-404.

N89
29784
UNCLAS

CU 508145
N89-29784

511-39
88
211952

# THE FORCE
# A Portable Parallel Programming Language Supporting Computational Structural Mechanics

November 18, 1987 Progress Report

**Principal Investigator:**
Harry F. Jordan (harry@boulder.colorado.edu)
**Research Assistant:**
Muhammad S. Benten (benten@boulder.colorado.edu)
**Associated Personnel:**
Juergen Brehm
Aruna Ramanan

Computer Systems Design Group
Electrical and Computer Engineering Department
University of Colorado
Boulder, Colorado 80309-0425

## Project Summary

This project supports the conversion of codes in Computational Structural Mechanics to a parallel form which will efficiently exploit the computational power available from multiprocessors. The work is a part of a comprehensive, Fortran-based system to form a basis for a parallel version of the NICE/SPAR combination which will form the CSM Testbed. The software is macro-based and rests on the "force" methodology developed by the principal investigator in connection with an early scientific multiprocessor. Machine independence is an important characteristic of the system so that retargeting it to the Flex/32, or any other multiprocessor on which NICE/SPAR might be implemented, is well supported. The principal investigator has experience in producing parallel software for both full and sparse systems of linear equations using the force macros, and other researchers have used the Force in finite element programs. It has been possible to rapidly develop software which performs at maximum efficiency on a multiprocessor. The inherent machine independence of the system also means that the parallelization will not be limited to a specific multiprocessor.

N89-29784

# THE FORCE LANGUAGE

- A Fortran based parallel programming language
- For shared memory multiprocessors

Flexible Computer Corp. Flex/32     Encore Multimax
Sequent Balance     Alliant FX/8

- Independent of number of processes
- Parallel execution of loops, cases, subroutines
- Synchronization: barrier named critical sections producer/consumer
- Management of variables:

|  | Shared | Private |
|---|---|---|
| (local) | × | × |
| Common | × | × |

- Dynamic creation of parallel work
- Efficient implementation of primitives
- Language description and manual available
- Shell scripts for compilation, execution

The Force[1] is implemented as a macro preprocessor on the Flex/32[2] multiprocessor located at LaRC. Performance of matrix multiply and Gaussian elimination were reported at the August 25, 1986 CSM Grants Review. The complete documentation is in the Force User's Manual[3] published as a technical report in October 1986 and most recently revised in October 1987.

The software consists of Unix stream editor scripts, macro definitions for m4, and Unix shell scripts to invoke the software and interface it to Flex/32 compilation and execution. The whole is a parallel extension to Fortran, compatible with systems on the Encore Multimax, Sequent Balance 8000 and Alliant FX/8 multiprocessors. Force programs can be run unchanged on any of the machines.

Dense matrix algorithms in the Force show close to linear speed up when run with from one to 18 processors. The maximum speed of matrix multiply was 1.1 MFLOPS while that for the more complex Gaussian elimination was 0.54 MFLOPS. Both were for unoptimized Force programs. Experience shows that higher speeds can be obtained by optimization techniques such as loop unrolling.

Efficiency concerns lead to careful analysis and redesign of existing macros, such as the thorough analysis and optimization of the barrier by Arenstorf[4].

## CSM Applications of the Force

- Parallelize RED module from SPAR's INV processor
    - Simple, low-level parallelization; no redesign
    - Speedup of 3+ on 8 Flex/32 processors

Other Groups:

- Gene Poole wrote Conjugate Gradient for Flex/32 in Force
- Charbel Farhat has numerous FE codes written in Force
    - Profile solver
    - Element by element computations
        - . Linear
        - . and Nonlinear
    - Preconditioned conjugate gradient
    - Block asymmetric factorization
    - Homotopy equations
    - Eigenvalue solver

The identical source for these has been run on:

- Encore Multimax
- Alliant FX/8
- Sequent Balance
- Cray 2

The Force on the Flex/32 supports parallelization of SPAR in the testbed system. Parallelization can be attacked two ways: confine parallelization to low level modules or redesign a parallel SPAR. The first was applied as a learning exercise, but as expected, did not yield large performance increases. For significant gain in performance, the whole program must be parallelized. This requires a thorough understanding of the structure of SPAR, which is not well documented. An analysis of data dependencies will be needed to implement SPAR on any multiprocessor.

The RED module from SPAR's INV processor was parallelized using the low-level approach which was primarily characterized by the replacement of sequential DO loops with parallel DOALLs. Without algorithm redesign, this simple approach yielded a speedup of just more than 3 using 8 processors of the Flex/32.

Others have used the Force to implement newly designed parallel algorithms for structural mechanics. One of the major users has been Charbel Farhat of the University of Colorado Center for Space Structures and Controls. Dr. Farhat has found the Force useful to write numerous finite element codes so that they can be run unchanged on several different multiprocessors.

# PROGRESS

## Since our last CSM Grants/Contract Review
## on August 25-26, 1987

- A new Force Manual has been published, including:
  - Askfor DO for dynamic work generation
  - Self-scheduled parallel case macro
  - Continuation lines
- The Force has been ported to the Cray 2
  - Simple test cases run correctly
  - Charbel Farhat has run substantial programs

    ...The last known bug has been corrected

A major improvement in the Force as a complete parallel programming language is support for dynamic generation of parallel work. While many scientific codes can be written without this capability, it is of use in adaptive and search type algorithms. It can be explicitly coded by the user in the original version of the Force, but this is an involved and error prone process. The Askfor DO macro was developed to provide a basic level of support for this capability without altering the structure of the language in any major way.

Efficiency of Force constructs has not only been improved by the revision of the Barrier mentioned previously, but also a new macro for parallel case execution has been introduced. By allowing the work to be self-scheduled instead of prescheduled, this macro can improve efficiency of the parallel case in certain contexts. Convenience of the system has been extended by allowing the Force macros to use continuation lines.

As a result of interest at our last grantees review, the Force has been ported to the Cray 2. The implementation has not been completely tested, but it runs simple test cases correctly and there are no known problems at this time. Some substantial codes written by Charbel Farhat have been run on the Cray 2 using the system, and that part of the Force which he uses seems to be correct.

## REFERENCES

[1] H. F. Jordan, "The Force," in *The Characteristics of Parallel Algorithms,* L. H. Jamieson, D. B. Gannon and R. J. Douglass, Eds., Chap. 16, MIT Press (1987).

[2] H. F. Jordan, "The force on the Flex: Global parallelism and portability," to appear in *Parallel Processing and Medium Scale Multiprocessors,* Arthur Wouk, Ed., SIAM, Philadelphia, PA, 1987.

[3] H. F. Jordan, M. S. Benten, N. S. Arenstorf and Aruna V. Ramanan, "Force User's Manual," *ECE Tech. Rept. 86-1-4R,* Computer Systems Design Group, Dept. of Electrical and Computer Engineering, University of Colorado, Boulder, CO, 80309-0425 (Revised Oct. 1987).

[4] N. S. Arenstorf and H. F. Jordan, "Comparing Barrier Algorithms," *ECE Tech. Rept. 87-1-2,* Computer Systems Design Group, Dept. of Electrical and Computer Engineering, University of Colorado, Boulder, CO, 80309-0425 (June 1987).

N89
29785
UNCLAS

N89-29785

iJ 535353
V 3127208

5/2-39
188
211953

C-4

# METHODS FOR DESIGN AND EVALUATION OF PARALLEL COMPUTING SYSTEMS
## (The PISCES Project)

Terrence W. Pratt
Robert Wise (MS candidate)
Mary Jo Haught (MS candidate)

Virginia Institute for Parallel Computation
Department of Computer Science
University of Virginia
and
ICASE
NASA Langley Research Center

## Summary

The PISCES project started in 1984 at the University of Virginia and ICASE under the sponsorship of the NASA CSM program. A PISCES 1 programming environment and parallel Fortran were implemented in 1984 for the DEC VAX (using UNIX processes to simulate parallel processes). This system was used at ICASE and the University of Virginia for experimentation with parallel programs for scientific applications and AI (dynamic scene analysis) applications. PISCES 1 was ported to a network of Apollo workstations by N. Fitzgerald.

N89-29785

# Long Term Objective

- Develop new methods for the EVALUATION of parallel computer architectures:
  - — For large-scale scientific software systems used by NASA
  - — Leading to the DESIGN of better parallel systems
- Problem areas:
  - — Variety of available parallel architectures
  - — Software layers impact performance
  - — Performance on SYSTEMS is more important than performance on individual programs
  - — Existing Fortran software base

## Long Term Objective

The long term objective of the PISCES project is to develop better ways to evaluate the new parallel computer systems that are coming on the market. The particular target is evaluation of the effectiveness of these parallel systems for the large- scale scientific software systems of interest to NASA. The ultimate goal of the work is to influence the design of the next generation of parallel systems to better meet the needs of NASA software systems.

The major problem areas in evaluating the emerging commercial parallel machines are several. First, the large variety of different parallel architectures available makes evaluation difficult. To reprogram several large NASA systems for each different architecture in order to evaluate performance differences would be expensive and time-consuming.

Second, evaluation of these machines should be in the context of the same sort of programming environments, using Fortran or similar high-level languages, that have traditionally been available on sequential machines. These software layers have an impact on performance that is important to evaluate.

Third, the performance of parallel machines on large NASA software systems is the most important basis for evaluation. Performance on small test programs or on particular algorithms is not of as much interest.

Fourth, the evaluation must take into account the amount of reprogramming of the existing NASA software base that would be required to effectively use these parallel machines. Most of this existing code is in Fortran.

# Approach: Experimental

- Build a parallel environment for scientific applications
- Carefully define the software-provided "virtual machine"
- Implement on a variety of machines
- Experiment with language designs (for new code)
- Experiment with porting large systems (for old codes)
- Measure performance differences

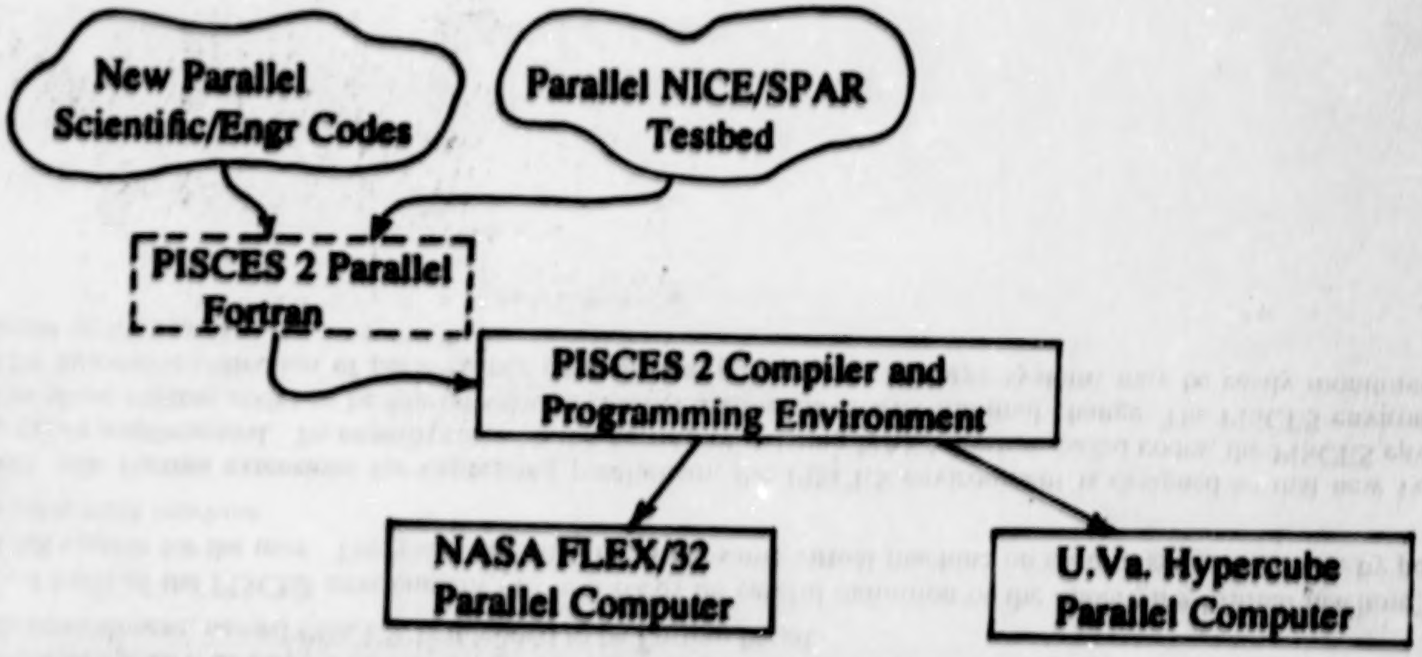
## Approach: Experimental

The PISCES project goal is to build a testbed programming environment to support the evaluation of a large range of parallel architectures. This environment, named PISCES, is intended to be Fortran based.

The conceptual basis of the PISCES environment lies in a focus on careful definition of the underlying "virtual machine" provided by the PISCES system for the user. The goal is to implement the same virtual machine on each target architecture, by porting the PISCES software to each machine.

To experiment with Fortran extensions for expressing parallelism, the PISCES environment is designed so that new Fortran extensions can be easily implemented. To experiment with the porting of existing NASA Fortran-based codes, the PISCES environment is designed to allow Fortran codes to be run on various parallel architectures with minimal change. The PISCES environment provides support for automatic collection of performance data, so that performance of large systems may be easily monitored and measured on different architectures.

# New Machine Independent Parallel Fortran Implemented on NASA FLEX/32


New Parallel Scientific/Engr Codes
Parallel NICE/SPAR Testbed
PISCES 2 Parallel Fortran
PISCES 2 Compiler and Programming Environment
NASA FLEX/32 Parallel Computer
U.Va. Hypercube Parallel Computer

**Result: New parallel algorithms and parallel versions of existing sequential codes may be easily evaluated on the FLEX/32**

## Summary of Progress

Major results of the PISCES project include an implementation for the NASA Flexible FLEX/32 parallel computer of the PISCES 2 parallel programming environment. PISCES 2 includes a set of extensions to Fortran for parallel computation.

The PISCES 2 system is fully operational on the CSM FLEX/32. The CSM NICE/SPAR testbed system is being modified for parallel operation. An implementation of the testbed in Pisces Fortran on the FLEX/32 is in progress.

The PISCES 2 system is being ported to an NCube/ten hypercube parallel computer at the University of Virginia. The NCube machine has 113 processors with sophisticated parallel graphics and parallel disk subsystems.

The PISCES 2 system on the FLEX/32 provides a useful software base for writing and evaluating new parallel Fortran codes and for modifying and evaluating existing codes for a parallel environment.

# Results

- Design of PISCES 2 parallel programming environment: COMPLETED
- Implemention on NASA FLEX/32: COMPLETED
- IN PROGRESS:
  - — NICESPAR testbed port
  - — New parallel algorithms
  - — Hypercube implementation

## Results

As noted on the previous slide, the design and implementation of the PISCES 2 system is complete and the system is installed on the CSM FLEX/32.

The port of the NICE/SPAR testbed to PISCES 2 is in progress. The port of PISCES 2 to the U.Va. NCube/ten hypercube is also in progress. Researchers at ICASE, Duke, and U.Va. are using the FLEX/32 system to implement and evaluate new parallel algorithms.

# PISCES 2: Main Concepts

- Architecture Independent Virtual Machine
- Multiple Grain Sizes of Parallel Operation
- Clustered Architecture
- Operating System = Static Set of Tasks in Each Cluster
- Dynamic User Task Initiation; Asynchronous Message Passing
- 'Forces' (a la Harry Jordan) with Shared Variables
- Programmer Control of Virtual Machine to Hardware Mapping

## PISCES 2: Main Concepts

The PISCES 2 system provides a rich environment for experimenting with parallel programming concepts. The main concepts are:

1. The virtual machine is relatively independent of the underlying architecture, so that programs in Pisces Fortran are not written using constructs peculiar to one parallel architecture.

2. The virtual machine provides several granularities of parallel operation. In PISCES 2 parallel operation is provided between large-grain "clusters of tasks" and "tasks within a cluster" and medium-grain loop iterations and arbitrary program segments.

3. The virtual machine architecture is "clustered", with each cluster representing a group of processing and memory resources of the underlying machine. The cluster organization is static within a single run, but it may be varied between runs.

4. The operating system is represented as a static set of tasks that run within each cluster of the virtual machine. User tasks invoke operating system functions by sending messages to these operating system "controller" tasks.

5. User programs are represented at the top-level by a set of tasks that are dynamically initiated and terminated during a run. Tasks communicate using asynchronous message passing. Message passing queues are infinite (to available memory), and the receiving task may "accept" a message in a variety of different ways, or never.

6. Many of the "FORCE" constructs developed by Harry Jordan are included in PISCES Fortran. These constructs provide medium-grain parallelism, including parallel execution of loop iterations, subroutine calls, and arbitrary program segments. Synchronization mechanisms include barriers and critical regions with lock variables. Communication is via shared variables in COMMON blocks.

7. The programmer controls the mapping of hardware resources to the "clusters" of the PISCES 2 virtual machine. Each run of a program may be configured differently. Currently the mapping consists of assignment of a group of processors to each cluster for running tasks and "forces".

# PISCES 2 Implementation

- Extensions to Fortran 77 for parallelism
- Preprocessor: Pisces Fortran ---> standard F77
  - In-line expansion of parallel constructs
  - Run-time library
- Configuration Environment
  - Configure virtual machine (VM)
  - Map hardware PE's to VM clusters
  - Create loadfiles to download
  - Choose run-time options: tracing, timelimit, etc.

## PISCES 2 Implementation

The PISCES 2 system as implemented on the CSM FLEX/32 consists of several software components:

1. *Preprocessor*. The extensions to Fortran 77 that form the Pisces Fortran language are implemented with a preprocessor that converts Pisces Fortran to standard Fortran 77, which is then compiled with the FLEX Fortran compiler. The preprocessor converts many parallel constructs to in-line Fortran code. The core parallel constructs are implemented with a small run-time library of routines called from the Fortran code.

2. *Configuration environment*. Before each individual run of a parallel program, the user works in the PISCES "configuration environment" to choose the configuration options for that particular run. These options include the choice of the number of clusters and the cluster numbering, choice of the mapping of hardware resources to these clusters, creation of loadfiles for downloading to hardware PE's, and choice of run-time options such as time limits, tracing options, etc. These options are chosen with a series of menus and prompts. The user does not need to know FLEX OS commands for this activity.

Configurations may be saved in files for later editing/reuse in other runs of the same or different programs. Thus a user can build a library of configurations for use in comparative performance evaluation studies.

# PISCES 2 Implementation (cont)

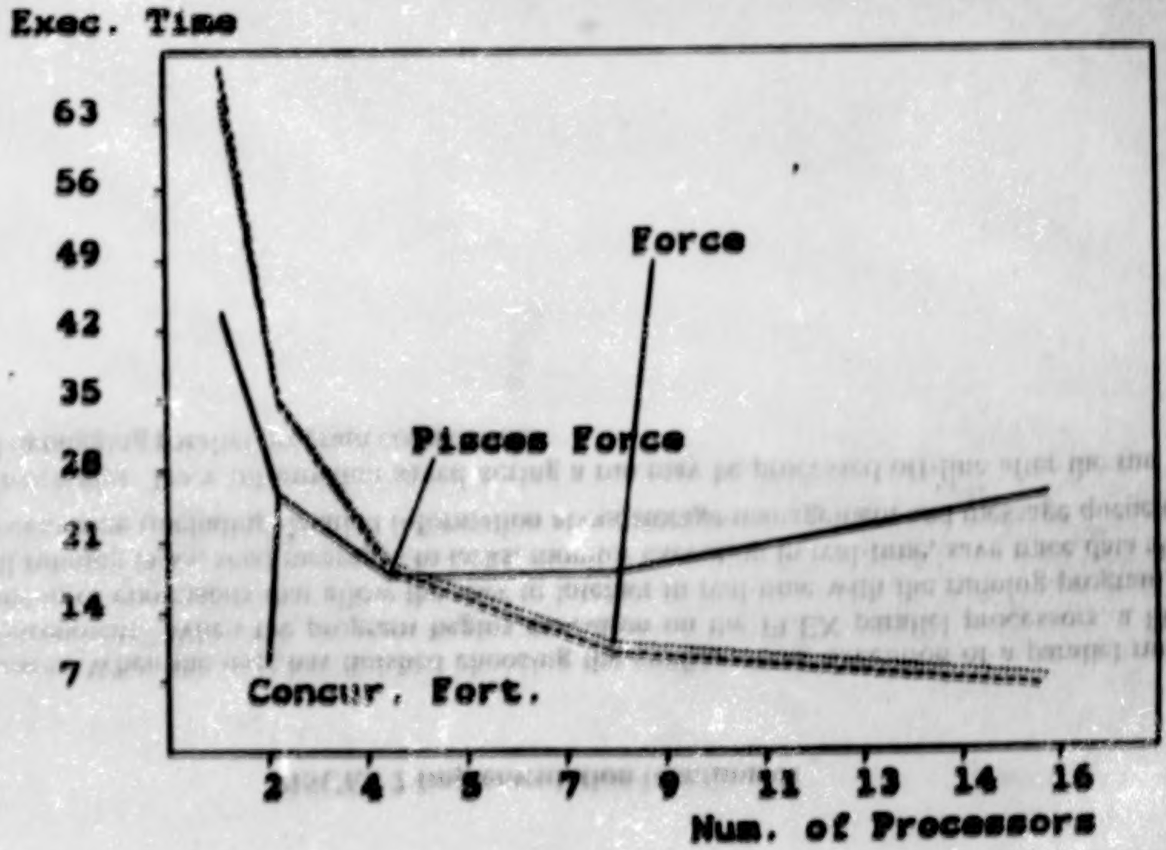
- **Execution Environment:**
  - — **Control execution: initiate/kill tasks, send messages**
  - — **Monitor execution: observe system state in real-time**
  - — **Trace options: save events/times in file or display**
- **Postprocessing: massage trace files**

## PISCES 2 Implementation (continued)

3. *Execution environment.* When the user has finished choosing the configuration, execution of a parallel run may be initiated from the configuration environment. When the program begins execution on the FLEX parallel processors, a PISCES "execution environment" provides a menu of commands that allow the user to interact in real-time with the running program. Options include the ability to initiate or kill running tasks, send messages to tasks, monitor execution in real-time, save trace data about key events in trace files, and view the system state (including detailed information about storage management and message queue contents.)

4. *Postprocessing of trace files.* Trace information saved during a run may be processed off-line after the run to obtain detailed information for timing and debugging parallel program components.

Comparison of Execution Times
For Factorization (N=300 Beta=75)



## Performance Comparisons

Conventional wisdom would suggest that machine dependent software provided by the manufacturer of a parallel system would provide better performance than systems such as PISCES 2 or Jordan's FORCE that are implemented as software layers on top of the vendor's software. These performance curves from a study by Prof. Merrell Patrick and Mark Jones of Duke University illustrate that this conventional wisdom can be wrong.

This graph from a Choleski factorization algorithm coded in Flexible's Concurrent Fortran, Jordan's FORCE, and Pisces Fortran shows that the software overhead in Concurrent Fortran quickly dominates this computation, while the software overhead in both PISCES 2 and Jordan's FORCE remains relatively low as the number of processors increases.

These performance measurements are preliminary. More extensive performance measurements and performance tuning of PISCES 2 are in progress.

## Publications and Presentations

[1] Pratt, T. "PISCES: An Environment for Parallel Scientific Computation," *IEEE Software*, 2, 4, July 1985. (Also ICASE Report 85-12, February 1985.)

[2] Fitzgerald, N. *Implementation of a Parallel Programming Environment*, M.S. Thesis, Computer Science Dept., U. of Virginia, August 1985.

[3] Patrick, M. and Pratt, T. "Communications Oriented Programming of Parallel Iterative Solutions of Sparse Linear Systems," *Communications in Applied Numerical Methods*, vol. 2, 255-261, 1986.

[4] Pratt, T. "Finding the Right Virtual Machine for Parallel Applications Programming," presentation at *Workshop on Performance Efficient Parallel Programming* (September 1986, Seven Springs, PA, sponsored by NSF and Carnegie-Mellon University).

[5] Pratt, T. "The PISCES 2 Parallel Programming Environment," presentation at *Parallel Languages and Environments Workshop* (November 1986, Williamsburg, VA, sponsored by ICASE and NASA).

[6] Pratt, T. "Short Course: Introduction to the PISCES 2 Parallel Programming Environment," presented at NASA LaRC/ICASE (March 1987) and U. of Virginia (April 1987).

[7] Pratt, T. *PISCES 2 User's Manual (Version 2)*, ICASE Interim Report 2, July 1987.

[8] Pratt, T. "The PISCES 2 Parallel Programming Environment," *Proceedings 1987 International Conference on Parallel Processing*, St. Charles, IL, August 1987, 439-445 (also ICASE Report 87-38, July 1987).

N89
29786

UNCLAS

N89-29786

S13-39
238
211954

T174 9667

See p. 303

# MULTIPROCESSOR ARCHITECTURE: SYNTHESIS AND EVALUATION

Hilda M. Standley

Department of Computer Science and Engineering
The University of Toledo
Toledo, Ohio 43606

Performance of a multiprocessor is determined by

- the algorithms
- the programming language
- the program
- the language support environment and operating system
- number of processing elements
- characteristics of the processing elements
- interconnection network
- shared memory organization

The difficulty in the analysis of multiprocessor performance may be attributed to the large number of factors that may affect performance both independently and through interactions. Such factors may be roughly divided into software and hardware categories: software--the applications algorithm, the nature of the programming language, the efficiency of the program, and the language support environment and operating system; hardware--the number of processing elements, the capabilities of the processing elements, the interconnection network, and the organization of memory.

Goals:

- Ignore the algorithm effect
- Remove the language/programming effect
- Study only those characteristics of the structure of the architecture.

**The goal of this study is to remove the influence of the choice of algorithm used for a particular application and to remove the effects of the high-level language and the efficiency of the program. The study concentrates on only those characteristics of the structure of the architecture. The "structure of the architecture" is defined to include those parameters that distinguish an architectural design at the diagram level. For example, the interconnection network plays an integral part in such a description while the capabilities of the individual processing elements, while crucial to the execution of the program, are not represented in the diagram.**

**Removing the language/programming effect:**

- **Express maximum amount of parallelism**
- **Data Flow Diagrams (operation level)**
- **Data Flow Diagrams (program module level)**
- **Partitioning and mapping of data flow diagrams**

A high-level language notation to express the maximum amount of parallelism is required to assist in removing the language/programming effect. The EASY-FLOW language, based on the data flow paradigm, offers a mechanism for expressing the data dependencies between program modules, down to the level specified by the programmer. These data dependencies are obstacles to parallel execution. Modules which are not related by data dependencies may be executed in parallel. The execution environment must include a mechanism for the partitioning and mapping of the resulting data flow diagrams.

Study the impact of the memory organization and the interconnection network

A queuing network mathematical model is developed for representing the effect of expanding separate shared memories into a system of memory hierarchies.

The two elements of the architectural structure selected for initial study are the memory organization and the interconnection network. A queuing network statistical model for a multiprocessor with shared memory is expanded to include a hierarchy of memory modules at each shared memory cluster.

**Model is based on an expanded GMI (General Model for Memory Interference)**

**Performance is measured as the expected number of busy memories.**

The shared memory hierarchy model is based on the General Model for Memory Interference (GMI) suggested by Hoogendoorn. Each processor cycles between a random access to a particular level within a memory cluster and a time interval in which internal computation is performed. Requests to the same memory cluster are queued at the cluster. Performance is measured by the expected number of busy memories.

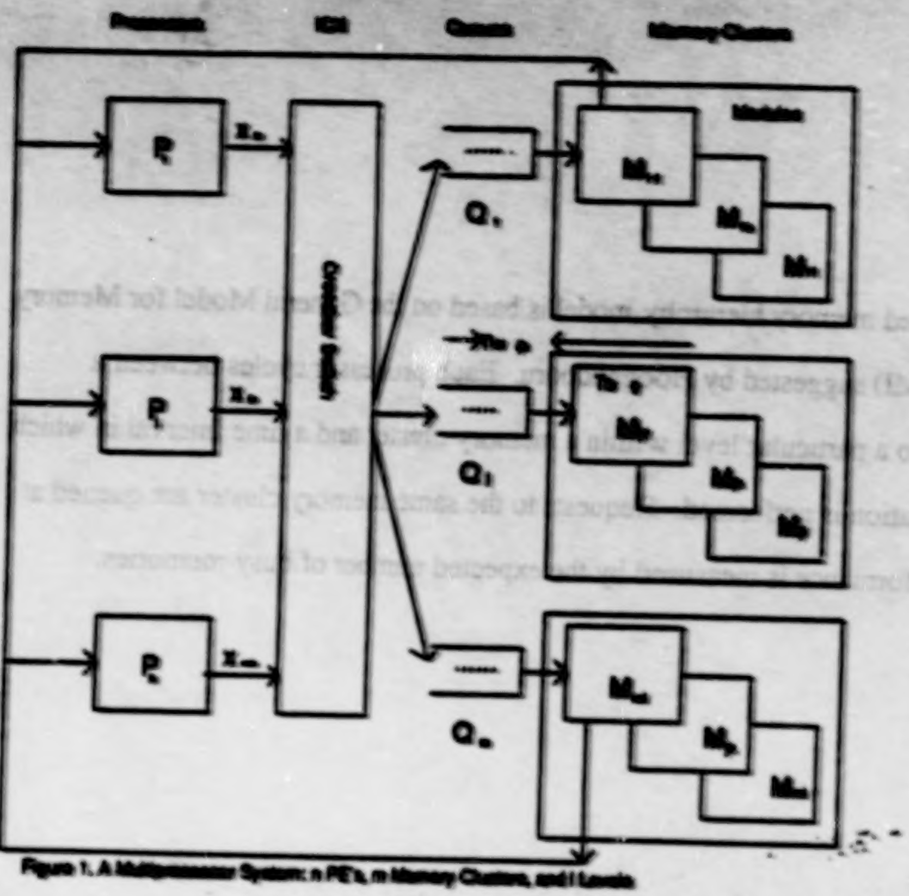

Processors
ICN
Queues
Memory Clusters
Modules
Crossbar Switch
Q
Q
Q

Figure 1. A Multiprocessor System: n PE's, m Memory Clusters, and l Levels

In the shared memory cluster multiprocessor model, the processors are connected to the memory clusters via a crossbar switch. It is assumed that this switch introduces no delay in accessing memory. Requests to memory are queued at each memory cluster. Delays in memory access time may be introduced by interference from other processors accessing memory levels within this same cluster.

A Network II.5 (CACI, Inc.) simulation has been developed in order to evaluate the analytic model. An eight-processor/eight-memory cluster system is evaluated under a variety of access distributions and intervals of computation time between requests to memory. The data collected from 63 simulation runs correlates with the results of the analytic model at 0.9950, overall.

## Modeling the effects of the interconnection network

A polynomial surface representation of performance is developed in a (k + 1) space.

Independent variables may be quantitative and/or qualitative:

- size
- average degree (per node)
- diameter
- radius
- girth
- node-connectivity
- edge-connectivity
- connection cost
- minimum dominating set size

For the analysis of the effect of the interconnection network on performance, a polynomial surface representation of performance is developed. Variables thought to influence the performance of a network are: size, average degree (per node), diameter, radius, girth, node-connectivity, edge-connectivity, connection cost, and minimum dominating set size.

**Performance measures:**

- message completion rate
- average message delay
- connection cost

Dependent measurements are used to gage performance. Typical performance measures are message completion rate, average message delay, and connection cost. Although the nature of the problem is for the different levels of the independent variables to determine a very much discrete set of performance points, the problem is viewed as being continuous in the performance variable.

## Optimization:

Response surface methodology (RSM) optimizes a response variable, based on some polynomial function of several independent variables.

Gradient vector may indicate direction of steepest ascent.

A polynomial function of several independent variables is used to estimate the performance surface. This function is estimated through curve fitting techniques. Response surface methodology (RSM) optimizes the response (performance) variable, working from this estimated polynomial function. In the situation where an optimum is not indicated, gradient vector methods may detect the direction of steepest ascent.

Figure 1.

[illegible]

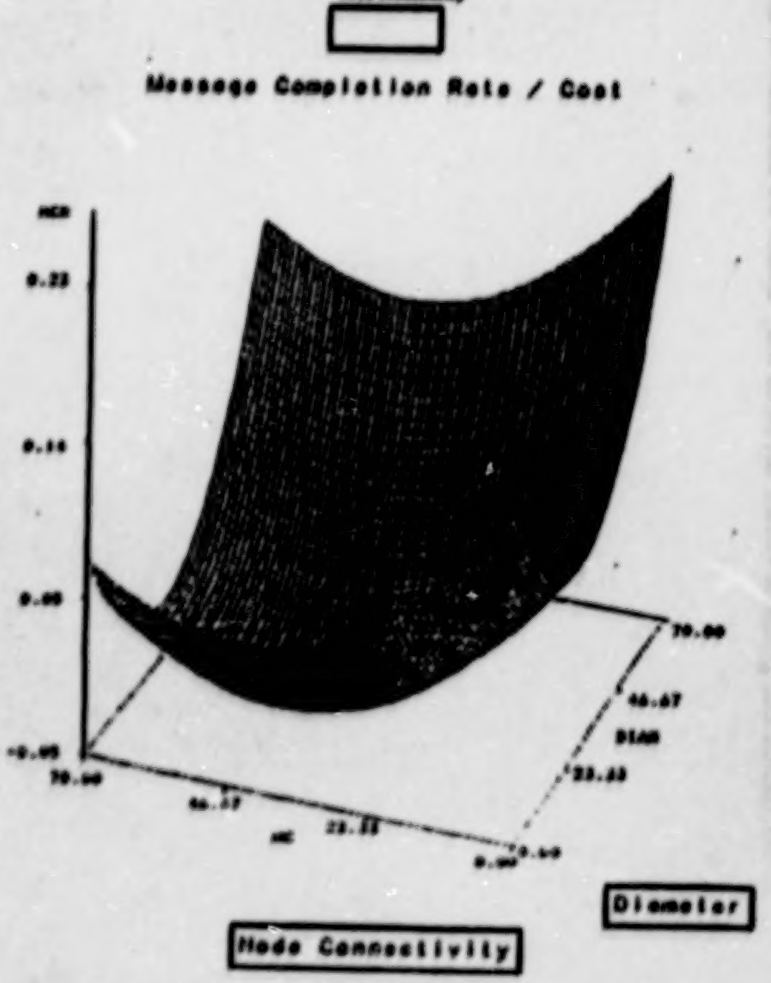
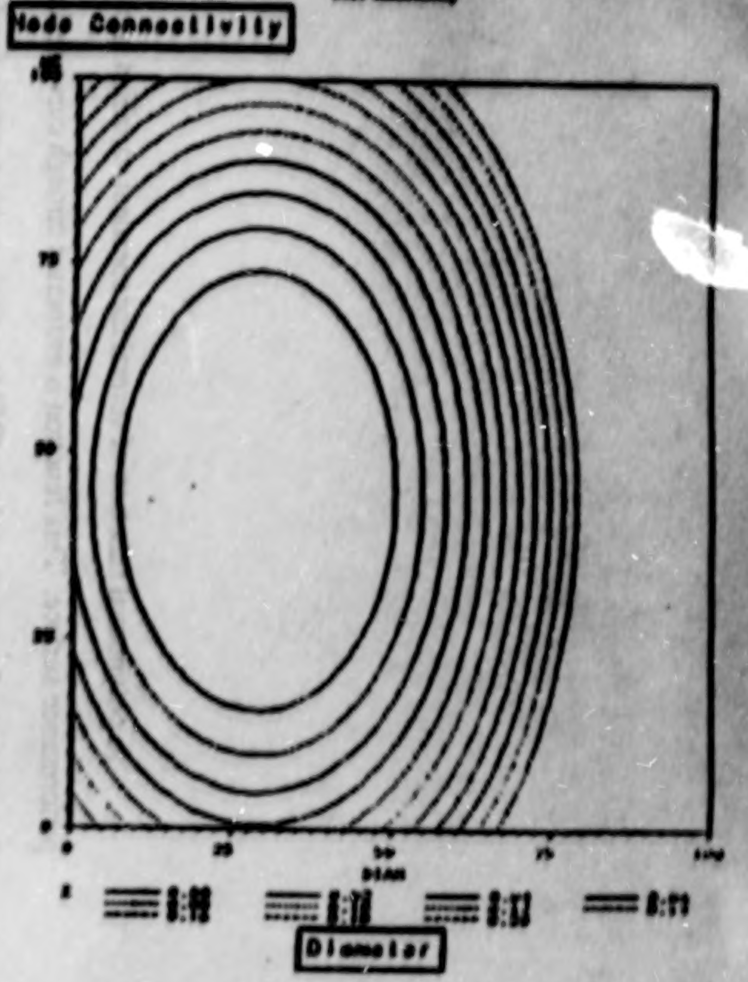


Figure 2.

[illegible]



ORIGINAL PAGE IS
OF POOR QUALITY

An application of this analysis uses independent variables of node-connectivity and network diameter and the performance measure of message completion rate/cost. It may be seen from the diagram that better network performance levels occur at the "corners" of the graph, for example when both diameter and node-connectivity are high.

# Network Synthesis

The results of this analysis may be used to identify appropriate levels of independent variables to indicate optimum or near-optimum performance networks. Existing, well-studied, networks; networks that are hybrids of existing networks; or completely novel networks may be suggested.

N89
29787
UNCLAS

N89-29787

34-39
328
211955

# Environmental Concept for Engineering Software on MIMD Computers*

L. A. Lopez and K. Valimohamed
Department of Civil Engineering
University of Illinois, Urbana, IL 61801

## Introduction

Novel computer architectures have been developed in an attempt to satisfy the performance specifications required to solve large nonlinear dynamic systems encountered in the sciences and engineering. In order to exploit the new architectures, it is often necessary to change the way we think about and write programs.

Several software options have been proposed to facilitate the programming of these sophisticated and complex machines [1-6]. However, existing solutions are rather restrictive or prohibitively expensive because they are not portable.

Recently, there have been some developments in environments and programming tools [7-10] that assist programmers in developing portable application code. However, these efforts have been restricted primarily to the control aspects of the program; i.e., describing to the computer when and how to do things in parallel. A key element in developing successful engineering systems is the management of large data spaces. Very little work has been done in this crucial area on MIMD machines.

The issues related to developing an environment in which engineering systems can be implemented on MIMD machines will be discussed. The problem is presented in terms of implementing the finite element method under such an environment. However, neither the concepts nor the prototype implementation environment are limited to this application. The topics discussed include: the ability to schedule and synchronize tasks efficiently, granularity of tasks, load balancing, and the use of a high level language to specify parallel constructs, manage data, and achieve portability. The objective of developing a virtual machine concept which incorporates solutions to the above issues leads to a design that can be mapped onto loosely coupled, tightly coupled, and hybrid systems.

---

[*] This work is supported by NASA Grant NAG-1-640

PRECEDING PAGE BLANK NOT FILMED

PAGE 322 INTENTIONALLY BLANK

N89-29787

**OBJECTIVE:**

**Define an MIMD Environment in Which We Can Implement Engineering Applications**

**CONSTRAINTS:**

**Solve Tomorrow's Problems as Opposed to Today's Problems Faster**

**Problems Will Have Large Data Spaces**

**Machine Independent Application Code**

**Portable Software Environment**

## OBJECTIVES AND CONSTRAINTS

The purpose of the project is to develop the concepts and to prototype a software architecture to support engineering systems on MIMD computers. A primary focus is to provide engineering programmers with a tool to express their problems without the need to become deeply involved in the concepts associated with parallel computers. The algorithm should transcend the computer architecture.

The governing philosophy of this research is to find methods to solve the largest problem possible within a given time *frame* rather than trying to solve existing problems faster.

The finite element method is the target application to be run on the environment. However, the support environment envisioned at this time is not limited to that application.

The kinds of problems envisioned for this environment are very large by today's standards. They will probably have data spaces in the 100 gigabyte range.

The investment in the next generation of programs will be so large that we will not be able to afford to start over with each new hardware concept. Therefore, both the application code and the environment will need to be machine independent.

## GUIDING PRINCIPLES:

1) Application Program Concept Transcends The Computer Architecture

2) Data Objects:

Are Primarily - Matrices and Hyper-Matrices

Are Organised - Primarily Hierarchically

3) Application Can Express Concurrency at Multiple Levels

- a) Vectorisation ( Typically Mfgr )
- b) Concurrency using Precompilers
  - i) Local software
  - ii) Mfgr Provides Compiler Directives
- c) High Level Language Layer (2 levels)

## GUIDING PRINCIPLES

The principles guiding us are:

> The application program concept transcends the computer hardware on which it executes. This is very important. One can express the finite element method independently of how the computer does the problem. This is not a new concept; FORTRAN statements are used to express algorithms rather than machine registers. Similarly, one can express the idea of natural parallelism in an algorithm independently of the method of achieving it.
>
> We propose that engineering programmers should be provided with tools to easily express various levels of potential parallelism in their algorithms without paying the consequences of fighting the details of implementation. It will be left to the combined hardware and software environment to decide what to do with the expressions.

To facilitate the data and memory management functions, the use of data objects will be necessary. Typical data objects will be matrices and hyper-matrices which are organised in a hierarchical form.

There are two types of parallelism to deal with - vectorization and concurrency. Vectorization is often handled automatically by the hardware and the compilers provided by the manufacturer. It should be transparent to the engineering programmer; we will assume its existence and will not deal with it directly in our research. Some researchers may argue that manufacturers do not do a good job of this. They will improve!

Concurrency is more difficult to detect, and is not performed automatically. There are several methods of specifying it in application systems today. A common solution is to use compiler directives. However, it is restricted to a particular manufacturer. An alternative is to use locally developed software like PISCES or the FORCE. They can then be ported from computer to computer without changing the application programs. Each of these methods is tied to the FORTRAN concept. Basically, they are sandwiched into what already exists.

An alternative is to develop a new high level language layer above FORTRAN. This is more difficult but eliminates the constraints of FORTRAN. Special constructs that are needed should be easier to implement. This method was used in the past in most large FEM (POLO/FINITE, DMAP/NASTRAN, DVS/ASKA, NICE/SPAR, ICES/STRUDL ...) systems. It is even more appropriate for MIMD computers.

## GUIDING PRINCIPLES (CONT. ):

**4) Application Programmers Should Express Potential for Concurrency - Not How to Do it Concurrently**

**They Should Not Have To:**

- **a) Assign Tasks to Processors**
- **b) Perform Inter-memory Data Movements**
- **c) Manage Memory or Data Directly**
- **d) Perform Explicit Synchronisation**

## GUIDING PRINCIPLES (cont)

Ideally, the user should only have to specify where the concurrency exists in the application program rather than explicitly defining how the application should be executed on a given machine. In the latter approach, for very large and complex problems, the application code tends to get "lost" in the details of: scheduling and synchronization of tasks; load balancing; inter-processor communication; managing the coherence of data objects in primary and secondary storage; and manipulating large data objects within a limited amount of memory.

Therefore, the programmer should not have to: assign tasks to processors; perform inter-memory data movements; manage memory or data directly; or perform explicit synchronization.

However, for improved performance, some interaction will be required from the programmer, the details of which will be described later.

## MAJOR PROBLEMS TO SOLVE AND AUTOMATE:

1) Load Balancing - Tasks Implied by the Programmer Cannot Translate Directly to Tasks on the MIMD Architecture. That Would Tie the Program to the Architecture.

2) Data Coherence - MIMD Architectures Protect Single Words at the Hardware Level. Our Basic Unit is a Matrix. We Must Provide the Coherence Checks.

3) Memory Management - Classical Structural Mechanics Does Not Mention Global Memories, or Distributed Memories, or Hybrid Memories. They Should Not Appear in the Expression of the Solution.

## MAJOR PROBLEMS TO SOLVE

Our objective is to develop concepts that hide most of the parallelism related activity from the programmer. In order to do this we have focused on solving (on paper) three major problems. They are automatic load balancing, data coherence, and data and memory management.

Automatic load balancing means that the programmer specifies only what can be done in parallel. The environment maps whatever is specified to the architecture in the best way possible by using the concept of relative granularity. Since the ideal task size for a processor will vary from one machine to the next, the tasks specified in the application program are interpreted and are executed in a manner which best achieves load balancing.

Data objects in large FEM systems are typically matrices and hypermatrices. Thus it becomes necessary for the the programmer, or in our case, the software environment to handle data coherence on large data objects. (Hardware data coherence is based on the granularity of a word size.) We believe that we can do this effectively via a combined queueing/data management/memory management system. Some of the ideas are clarified on the following slides.

Memory management has always been a problem for FEM programmers. The data space always exceeds the available memory (some undiscovered fundamental law of physics?). Memory management is not part of the classical FEM problem; neither is the concept of distributed memories; they should not explicitly appear in the FEM program either. Memory management must be handled automatically by the environment. In that way the programmer can concentrate on the conceptual FEM algorithm.

## METHODS OF SOLUTION:

**Are Evolutionary:**

- **Precompilers ( PISCES/FORCE )**
- **Libraries ( SCHEDULE )**
- **High Level Language Layer ( THAT'S US )**

## METHODS OF SOLUTION

The proposed solutions to problems relating to parallelism have been evolutionary, not revolutionary. In the past, FEM developers have used computer science concepts to develop environments with specific objectives. Precompilers were used in ICES. They are now being used in PISCES etc.. Libraries of routines called explicitly by the programmer were part of ASKA. They are also part of SCHEDULE - a system developed by Dan Sorensen et al at Argonne. High level languages were used in NASTRAN, POLO/FINITE, and NICE/SPAR and are proposed as a solution for the MIMD environment concept.

## PROPOSED SOLUTION:

**High Level Language Layer above Fortran**

- **a) Define Data Structures**
- **b) Define Processes for Operating on Data Structures**
  - **1) Simple Control Structures**
  - **2) Express Parallelism at Gross Level**
  - **3) Express Relative Granularity**
  - **4) Reference Data Objects and Corresponding Operations**

## PROPOSED SOLUTION

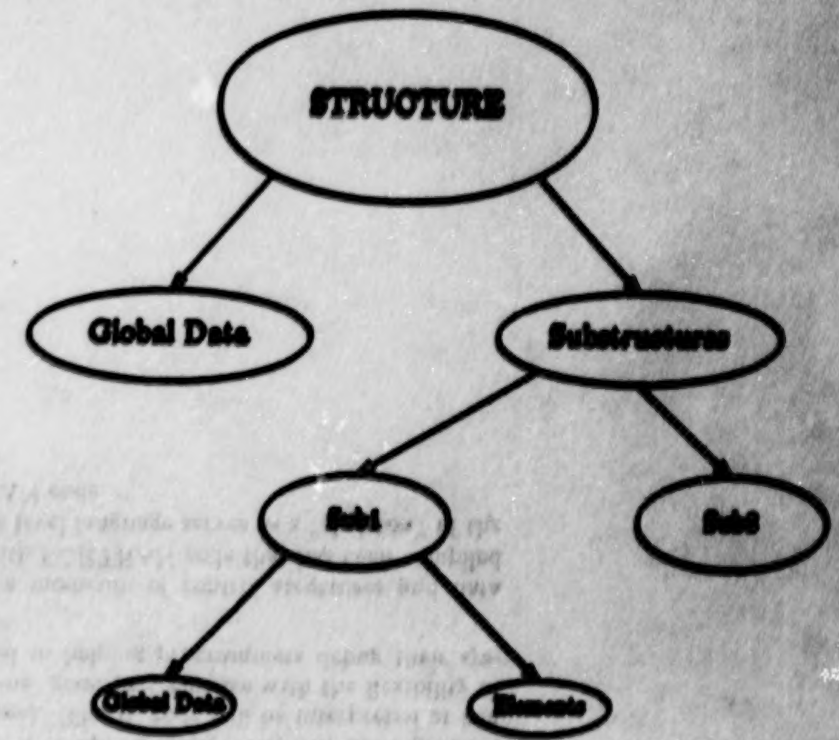
The combined machine independent automatic scheduling/DBMS/memory management proposed herein cannot be easily done by simply inserting additional commands into FORTRAN. Therefore we are investigating a language layer above FORTRAN that works in conjunction with FORTRAN. In that way we can provide a large number of features relating to large grain concurrency while taking advantage of whatever the manufacturer provides for fine grain concurrency and vectorisation at the FORTRAN level. The latter will probably be supported with tools already available or under development at CRM.

In the high level language, programmers will define data structures for data bases, and procedures that operate on the data. The system will utilise a compiled interpreter; i.e., the high level commands will be compiled to an IL (intermediate language). The IL code will be interpreted at run time. This technology has been used in the past to combine good performance with the flexibility of using an interpreter. The latter will prove very beneficial in helping programmers debug their systems, and in providing a mechanism for collecting statistics.

Interpreters can be slow. The idea is to provide a medium of control structures and data referencing in the high level language and to combine it with FORTRAN code that has been compiled for the target computer. Using the analogy of [6], the high level language serves as a "skeleton" of the application program while the "guts" consists of FORTRAN code.

## EXAMPLE OF HIGH LEVEL DATA DEFINITION:

```
STRUCTURES  relation  NUM_STRUCTURES
  attributes
   NUM_ELEMENTS
   NUM_NODES
   GEOMETRY     matrix        3  NUM_NODES
   TOPOLOGY     inverted_list    NUM_ELEMENTS
                    NUM_ENTRIES
   STIFFNESS    hypermatrix ...

   SUBSTRUCTURES   relation  NUM_ELEMENTS
    attributes
     NUM_ELEMENTS
     NUM_NODES
     GEOMETRY    matrix    3  NUM_NODES
       .
       .
       .
```


STRUCTURE
Global Data
Substructures
Sub1
Sub2
Global Data
Elements

**EXAMPLE OF DATA DEFINITION**

This slide demonstrates how a programmer might convert his concepts in data structure to something a computer could operate on. It is an example of how the DDL (data definition language) will appear to the programmer. Note that it is array structured and contains high level objects like relations, inverted lists, matrices, and hypermatrices.

## EXAMPLE OF CONCURRENT TASK EXPRESSION:

```
PARALLEL_DO for s = 1,n_sub_structures GRANULARITY Coarse

 SET_SUBSTRUCTURE (substructure_list( s, sub_structure:rg ) )

 PARALLEL_DO for i = 1,n_elem  GRANULARITY Medium

   CONNECTIVITY ( substructures(sub_structure,incidences (i)):rg, loc )

   PARALLEL_DO for j = 1,n_sub_matrices GRANULARITY Fine

     PUTMATRIX ( stiffness (sub_structure, loc(j)):rw:gd, elstiff(i,j):rd )

   END_DO
   .
   .
   .
```

## CONCURRENT TASKS

This slide demonstrates how a programmer might express a small part of his algorithm. Note the multiple levels of parallel "DO" loops. The PARALLEL_DO combined with the relative GRANULARITY implies both desire for concurrency and the programmer's conception of the size of the tasks in the loop. The granularity statement is a concession - the programmer is getting into the details of parallelism. However, we think it is a necessary evil. It will permit the task scheduler to make intelligent decisions about how to execute this task structure on a given architecture. In order accomplish this, the system environment will use data that describes the configuration of the machine. This information is provided when the environment is initialized on a given computer.

In addition, this slide shows a number of data references. Some are followed with a sequence of one or more colons(:) and some additional data. The latter is optional (but useful) information that the programmer provides to help the system make intelligent decisions. For example a :r implies the data is being used in a read only mode. Other tasks may use it too. If it is :w it implies a need for write access. If neither is given, it defaults to write; the latter is safe but may lead to poor performance. For instance, a given iteration of a parallel do loop can be a task. Write access to an object will hold the object until the end of the task unless the programmer provides an :i for immediate release. For instance, a reference such as SUB( data:w:i ) implies write access for the duration of the SUB operation, rather than the duration of the task in which the SUB operation is specified. It is similar to the fetch and add operation at the word level.

Each of the above attributes expands what the programmer needs to know about parallel computation. This is an admission of our failure to totally hide the parallel computational aspects of the problem. However, each attribute provides a simple mechanism to significantly improve performance.

## IMPLEMENTATION CONCEPT:

**All Data Definition is Compiled to an Internal Format**

**All High Level Processing Statements and Corresponding Data Object References are Compiled to an IL Code**

**A Copy of the System Resides on Each Processor. Using Monitor Concepts Each Processor Can Assume Master Status.**

## IMPLEMENTATION

As noted earlier, the high level language layer is most appropriate to meet all of the needs. Furthermore, interpretive data base management is much too slow to be practical. Consequently, all high level code and DBMS instructions will be compiled into IL code.

Data structures will be defined by the programmer using a high level language. The definition will be compiled into an internal format. This will allow the data manager to manipulate and examine data objects efficiently at run time.

There are a number of ways to implement parallel systems. After talking to individuals here and at Argonne we have concluded that a self-scheduling approach is most appropriate. In this scheme, each processor obtains 'work' (virtual process) from a global queue. This results in a dynamic load balancing scheme. Each processor will have access to the system functions. Using a monitor type approach in a shared memory machine, any processor can assume the responsibility of placing virtual processes onto the queue and removing them from the queue. Hence, one processor will not become a bottleneck by handling all messages and the queue. In a distributed memory architecture, the object that contains the monitor information can be passed around the network. A copy of the system resides on each processor.

In our original concept we had assumed that the number of processors available to a job would vary during the job; the system environment would obtain or release processors from the operating system as the work load varied in the job. However, most operating environments available today cannot support this. Consequently, the maximum number of processors must be specified when the job is initialized.

**RUN TIME SUPPORT SYSTEM:**

**Is Invoked Automatically at Two Levels**

- **When Explicit Parallel Constructs are Encountered**
- **When Data Objects are Referenced**

**Maps Relative Granularity to Absolute Using Config Tables for this Architecture**

**Does Process Control and Load Balancing Via its Own Virtual Process and Sleep Queues**

**Uses a Segmented Multi-level Virtual Memory Manager to Handle All Three Types of Memories**

**Can be Implemented in Extended FORTRAN Environments like PISCES, SCHEDULE, or FORCE.**

## RUN TIME SUPPORT SYSTEM

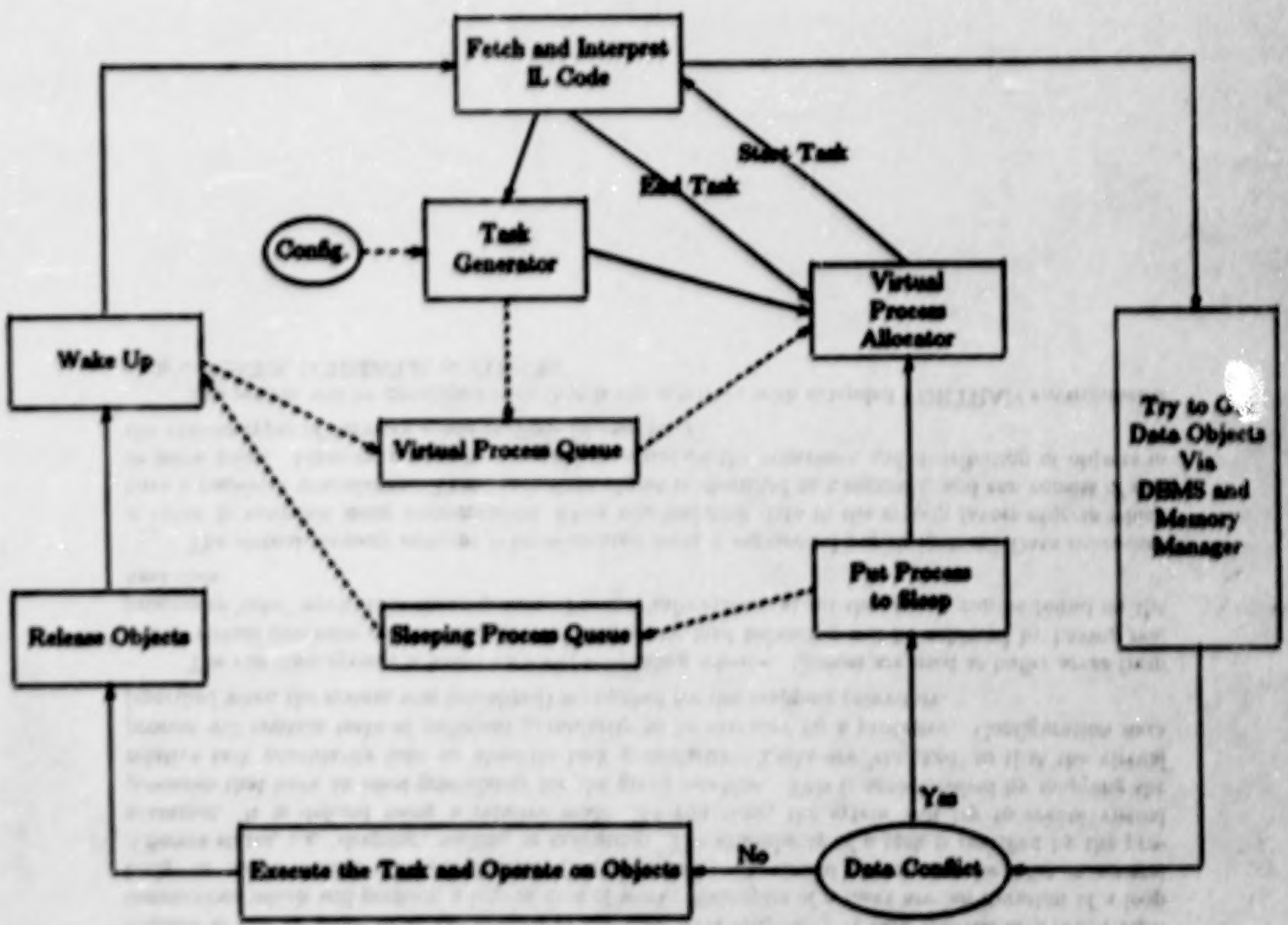
The run time support system is based on a virtual machine/virtual data base concept. It is invoked at two levels: when parallel constructs are encountered in the high level language; and, when data objects are referenced.

Virtual processes are created and then distributed among the real processors. A virtual process consists of one or more tasks (as defined in the high level language). A task consists of a set of HLL instructions which will perform a logical unit of work. Examples of a tasks are: an iteration of a loop body; or the execution of a subprogram (see next slide). A virtual process can be exist in several different states, e.g. 'sleeping', waiting or executing. The granularity of a task is specified by the programmer. It is defined using a relative scale. At run time, the sytem will try to create virtual processes that have an ideal granularity for the given machine. This is accomplished by mapping the relative task granularity into an absolute task granularity. Tasks are 'chunked' so that the virtual process will contain tasks of sufficient granularity to be executed on a processor. Configuration data (specified when the system was initialised) is required for the mapping procedure.

The run time system is based on a self scheduling scheme. Queues are used as buffer areas from which virtual processes can be manipulated. Dynamic load balancing will be achieved by having real processors 'take' work from these queues. Further information about the queues can be found on the next slide.

The virtual memory concept is implemented using a segmented-paging system. Data coherence is easier to maintain using segmentation, while manipulating data in the system favors objects which have a page-size granularity. Thus, each data object is identified as a segment, and can consist of one or more pages. Memory managers are used to maintain the coherence and distribution of objects in the various types of memory configurations (shown later).

The system will be developed such that it can interface with extended FORTRAN environments such as PISCES, SCHEDULE, or FORCE.


Fetch and Interpret IL Code
Start Task
End Task
Config.
Task Generator
Virtual Process Allocator
Wake Up
Virtual Process Queue
Try to G[illegible] Data Objects Via DBMS and Memory Manager
Put Process to Sleep
Release Objects
Sleeping Process Queue
Yes
No
Execute the Task and Operate on Objects
Data Conflict

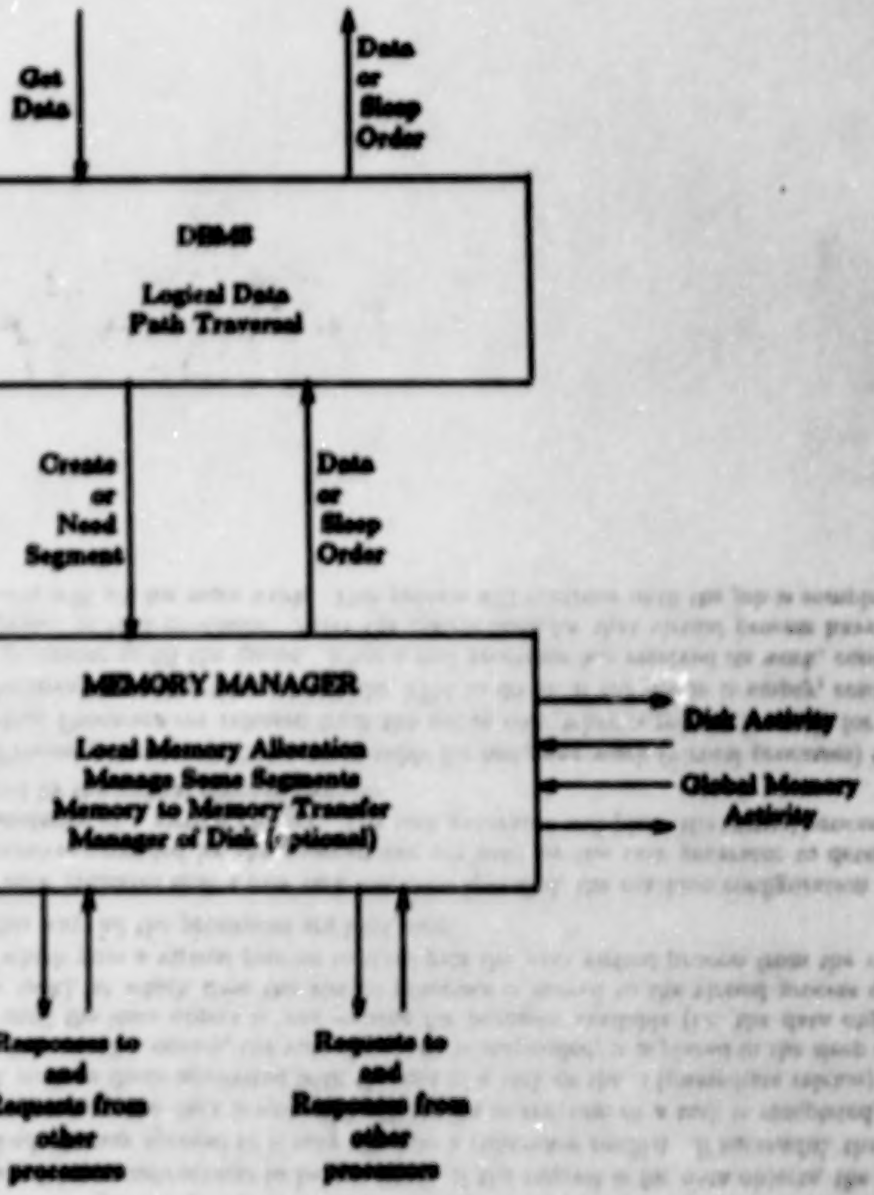
SYSTEM OPERATION

## SYSTEM OPERATION

This slide shows the general parts of the run time system that resides on each machine. The interpreter at the top fetches instructions to be executed. If the request is for data objects, the block on the right is invoked. It may succeed or it may run into a coherence conflict. If successful, the HLL instruction in task requesting the data is executed. When an instruction or a task is completed, data objects are released, such as those associated with the end of a task or the *ri* (immediate release) mentioned earlier. If a data conflict occurs, the virtual process is suspended, it is placed in the sleep queue and remains there until the data object it was waiting for becomes available (i.e. the data object is released by another task), at which time the virtual processes is moved to the virtual process queue. The real processor which puts a virtual process to sleep gets the next virtual process from the virtual process queue. In this way, all the processors are kept busy.

When the IL code indicates that a new task has been specified, the machine configuration information and the directives provided by the programmer are used by the task generator to determine the appropriate granularity of a virtual process. The task generator will place the virtual process in a queue which is shared by the virtual process allocator.

The Virtual Process Allocator (VPA) is responsible for assigning work (virtual processes) to the real processors. Virtual Processes are released from the queue only when a request is made for some work and the synchronization instructions permit the *VPA* to do so. If the queue is empty, control is passed to the task generator to fill the queue. After a real processor has received its work, control is passed to the interpreter of that processor. After the instructions for that virtual process have been executed, the processor will ask for more work. This process will continue until the job is completed.


Get Data
Data or Sleep Order
DBMS
Logical Data Path Traversal
Create or Need Segment
Data or Sleep Order
MEMORY MANAGER
Local Memory Allocation
Manage Some Segments
Memory to Memory Transfer
Manager of Disk (optional)
Disk Activity
Global Memory Activity
Responses to and Requests from other processors
Requests to and Responses from other processors

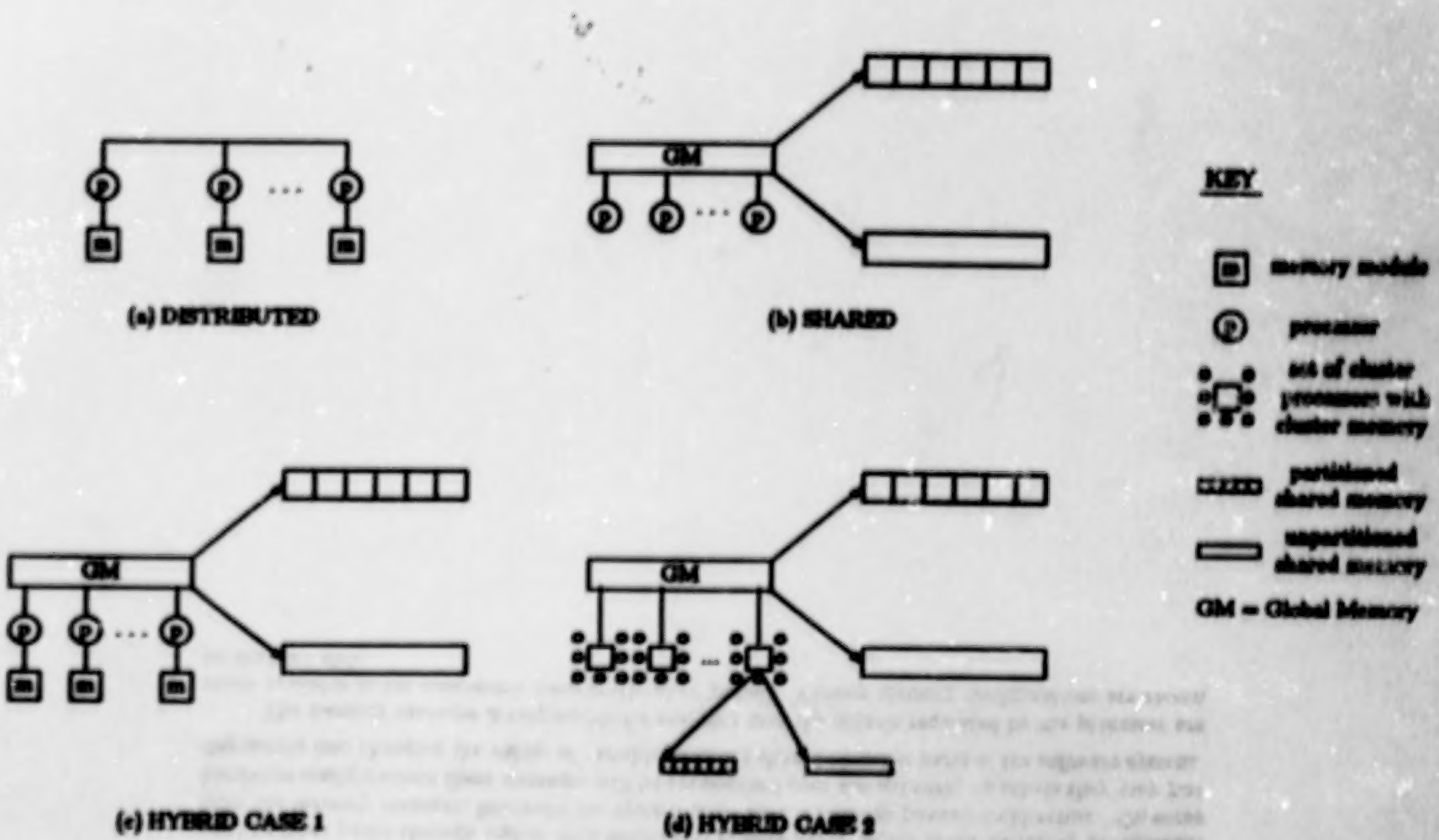
DBMS/MEMORY MANAGER INTERACTION

## OBJECT AND MEMORY MANAGEMENT

The data and memory management aspect is the most significant part of the system. It is what makes this concept totally different from other approaches; it makes parallel processing a virtual problem.

This slide shows that there are two major parts involved. The DBMS/IL code instructions are used to trace paths through logical data tables and objects. Each object in the hierarchy is requested from the memory manager. Internally the system must have a message passing architecture. On some hardware configurations these messages will be transmitted over the network; on others they may just degenerate into changing the values of variables that are shared by other parts of the software system.

The memory manager is responsible for ensuring that the objects requested by the processor are made available in the processor's memory (local or global). Various memory configurations are shown on the next slide.


KEY
m memory module
p processor
set of cluster processors with cluster memory
partitioned shared memory
unpartitioned shared memory
GM = Global Memory
GM
p
m

(a) DISTRIBUTED

(b) SHARED

(c) HYBRID CASE 1

(d) HYBRID CASE 2

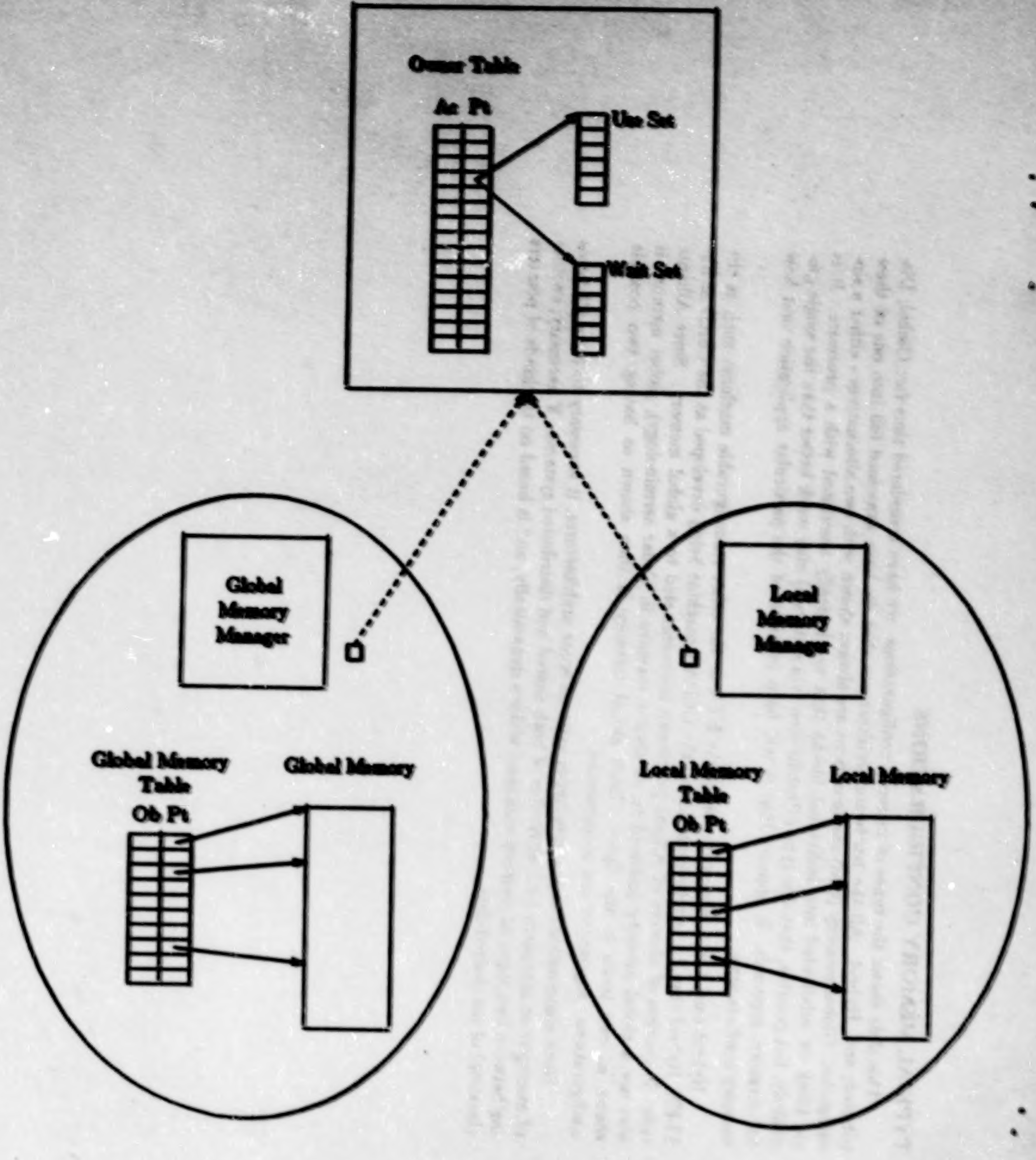
MEMORY CONFIGURATIONS

## TYPICAL MEMORY CONFIGURATIONS

This slide shows the types of memory configurations we have considered thus far: Global; Distributed; and, Hybrid. All the hardware architectures that we have examined fall into one of these categories. Global memory (GM) configurations are always shown with two alternatives - either a single block or subdivided into individual blocks that are logically associated with a processor. It is unlikely, but possible, that this type of subdivision (via *software*) may work better than the single global memory approach. It appears that it will be a function of the particular application and how memory conflicts arise.

Hybrid case 1 is a representation of the ETA or the other reconfigurable machines such as the FLEX. Hybrid case 2 is a representation of the CEDAR machine being developed at the CSRD at Illinois. It consists of clusters of Alliant processors interconnected to a global memory. Since Alliants also use a global memory (referred to as *cluster memory* in Cedar terminology), global memory is shown at two levels in the figure. Each global memory is then shown as having two possible configurations - segmented and unsegmented.

Since combinations of memory types exist on some architectures, it necessary to have all types of managers or managers with attributes of both shared and distributed systems. If necessary, switching between two types of memory managers is done dynamically, and is based on the kinds of pointers (location) of the desired data.


Owner Table
Ac Pt
Use Set
Wait Set
Global Memory Manager
Global Memory Table
Ob Pt
Global Memory
Local Memory Manager
Local Memory Table
Ob Pt
Local Memory

Memory Management Tables

## MEMORY MANAGERS

Memory managers insure that the desired object is present in memory for a processor when it needs it. There are two basic types of memory managers. One manages the memory on the processor itself. This system, called a local manager, sees its own memory directly. It controls what is in it, and where to put things in it. It needs no permission to look at data etc... The one exception is when its memory is used (partially) to store global tables. That will occur in totally distributed systems; in which case any information that is global to all processors is stored on one processor's memory.

The other type of memory manager that resides on all processors is a global manager. It is responsible for operating on either of the two types of global memory shown earlier.

Note that in the CEDAR machine there are two levels of global memory managers. Those at the cluster level have some of the attributes of the distributed memory managers - since clusters are distributed. Configuration tables will be used to determine which types of memory managers are to be used for a a given configuration.

Memory management for data objects is accomplished via ownership tables shown at the left. Each data object currently in use has an entry in that table. We estimate that, for the 100 gbyte problems, this table will contain about 5 megawords. It will be distributed evenly on a distributed memory system and stored in one memory on a global memory system. Since every object will not be in use at any given time, the full size of the table will not be required.

In the owner table we have access locks, information about which processes are currently using the object, and which processes are waiting for the object. (See slide on system queueing.)

There are some interesting problems with maintaining the use and wait sets. These result from our attempt to hide parallelism from the programmer. It is not always clear what he had in mind. In some instances combinations of entries are clear errors. In other instances they may be a result of the "system working ahead" or latency in the network; i.e., allowing multiple tasks to begin when it is known that there will be conflicts that can be resolved dynamically. If we don't allow this kind of activity, there will be unnecessary constraints on the application that may lead to serious degradation in performance. The :i flag in the high level language was introduced as a way for the programmer to inform the system memory manager that the apparent conflict was anticipated. All others will generate warnings.

**FUTURE:**

**Continue to Refine the Concept**

**Begin Implementing a Prototype**

## FUTURE

We will continue to refine the model. As we find problems, we will attempt to determine if they are a result of hiding too much of the parallel problem from the programmer. If so we extend the model of what a programmer sees in the high level language. If it is simply a mechanical problem related to the system design, or to some new type of architecture (we think we can handle chordal rings) we make the appropriate additions to our low level system concept.

This Spring we expect to begin a prototype of the memory management system. It will be something that we can use in simulations while the higher level components are developed. It may also be of use to others in the CSM family.

## REFERENCES

[1] Almasi, G.S., "Overview of parallel processing", *Parallel Computing,* Vol 2 (1985) pp 191-203

[2] Hockney, R.W., "MIMD Computing in the USA - 1984", *Parallel Computing,* Vol 2 (1985) pp 119-136

[3] Gajski, D.D. and J-K Peir, "Essential Issues in Multiprocessor Systems", *IEEE Computer* Vol 18, No 6., June 1985. pp 9-27

[4] Kuck, D.J. et al "Parallel Supercomputing Today and the Cedar Approach" *Science,* Vol 231, Feb 28 1986, pp 967-974.

[5] Snyder, L. "Type Architectures, Shared Memory and the Corollary of Modest Potential" *Department of Computer Science, FR-35* University of Washington. Seattle, WA 98195 TR 86-03-04

[6] McGraw, J. and T.S. Axelrod, "Exploiting Multiprocessors: Issues and Options" *UCRL-91734 preprint* Lawrence Livermore National Laboratory Oct. 31 1984

[7] Jordan, H.F., M.S. Benten, and N.S. Arenstorf, "Force User's Manual" *Dept. of Computer and Electrical Engineering, University of Colorado* October 1986

[8] Pratt, T.W., "PISCES: An Environment for Parallel Scientific Computation" NASA-ICASE Rep. No. 85-12

[9] Dongarra, J. and D. Sorensen, "SCHEDULE: Tools for Developing and Analyzing Parallel Fortran Programs," *Tech. Memo. 86, Argonne National Lab.,,* November 1986

[10] Padua, D.A., V.A. Guarna Jr. and D.H. Lawrie., "Supercomputing Environments" *CSRD Report No. 679* Center for Supercomputing Research and Development, University of Illinois, Urbana, IL. June 1987

N89
29788

UNCLAS

N89-29788

5/5-39
318
211956

AM351973

# HIERARCHIAL PARALLEL COMPUTER ARCHITECTURE DEFINED BY COMPUTATIONAL MULTIDISCIPLINARY MECHANICS*

**Joe Padovan**
**Doug Gute**
**Keith Johnson**
**University of Akron**

---

*Work supported by NASA Lewis and monitored by Chris Chamis

N89-29788

## GOAL

Develop Architecture for Parallel Processor Enabling Optimal Handling of Multidisciplinary Computation of Fluid-Solid Simulations Employing Finite Element and Difference Schemes

# Paper Outline

1. Goals
2. Paper Overview
3. Philosophical Directions
4. Modeling Directions
5. Static Poly tree
6. Dynamic Poly tree
7. Example Problems
8. Interpolative Reduction
9. Impact on Solvers
10. Summary
11. Future Directions

## Philosophical Thrusts

1. Reduce Load Per Processor
2. Reduce Number of Processors
3. Reduce I/O Between Processors
4. Provide for Most Natural Route of I/O Flow Between Processors
5. Enable Optimal Handling of Model Topology
6. Enable Optimal Handling of Automatic Mesh Refinement
7. Provide Logical Framework to Implement Generalized Saint Venants Type Model Reduction

# Modelling Directions

1. Static Single Level Models
   (Traditional Simulation)

   - Modelling Requirements Defined Initially

   - No Changes Occur During Computation

2. Dynamic Multilevel Models

- First Level of Model Refinement Established by User

- Multilevels of Refinement Established Via Automatic Physical Criteria ie.
  - Cavitation
  - Plasticity (Inelasticity)
  - Shock Formation
  - Flow Separation
  - High Stress and Strain Gradients
  - Etc.

# STATIC POLY TREE PARALLELISM

## Steps

1. Static Model Organized into Convenient Substructural Components

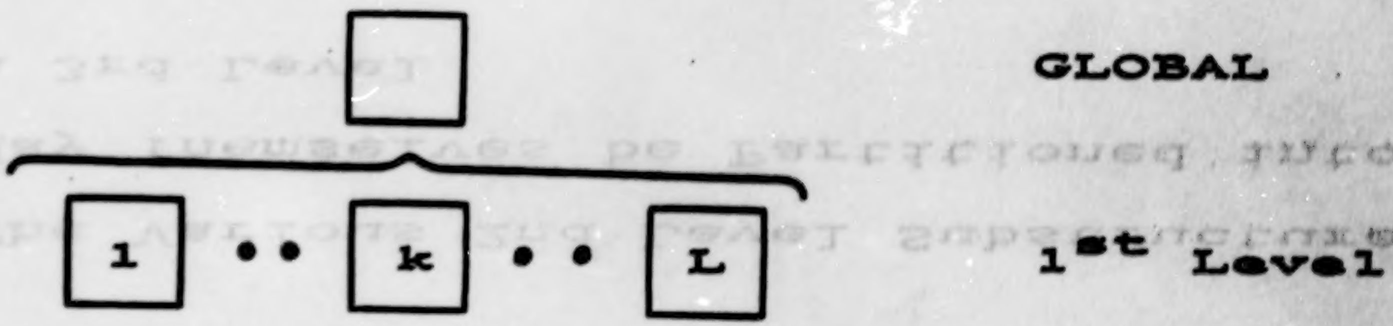
2. Each Substructural Component is Partitioned into Optimal Number of 2nd Level Substructures

3. The Various 2nd Level Substructure May Themselves be Partitioned into a 3rd Level

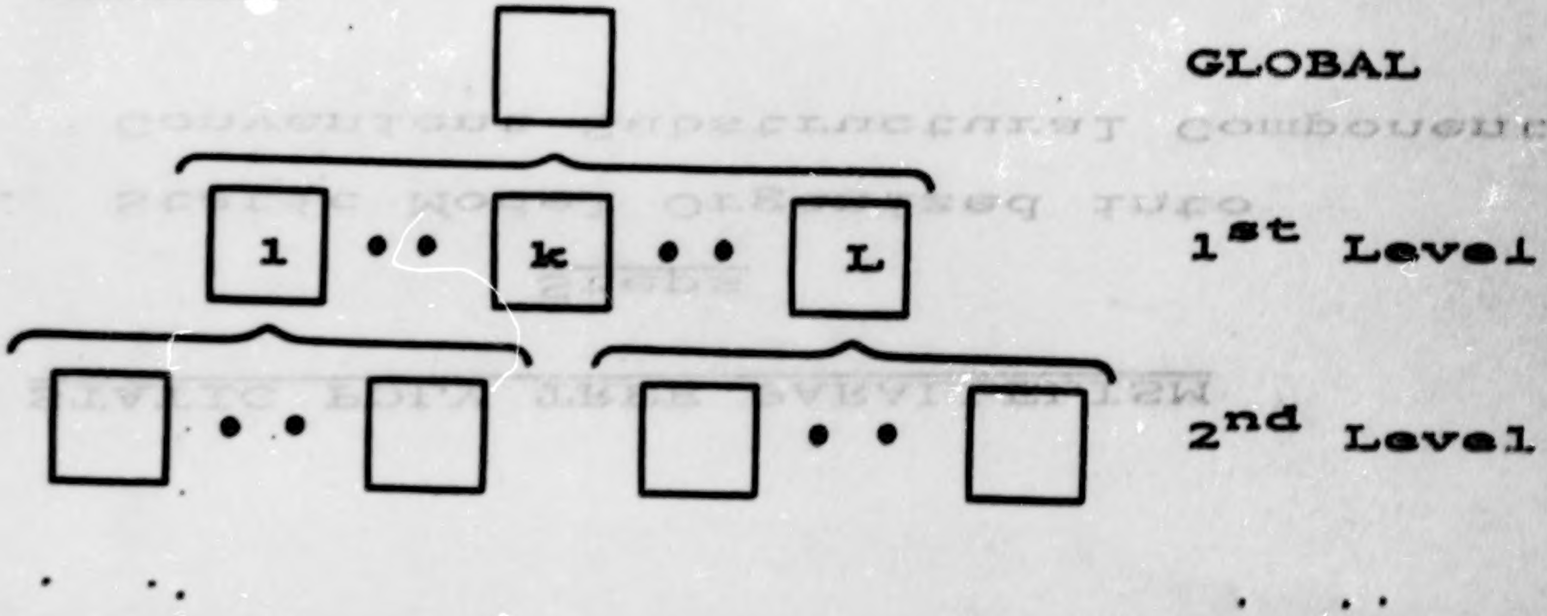
4. The Process May be Repeated to Yield a Multilevel Poly Tree

# DEVELOPMENT OF STATIC POLY TREE

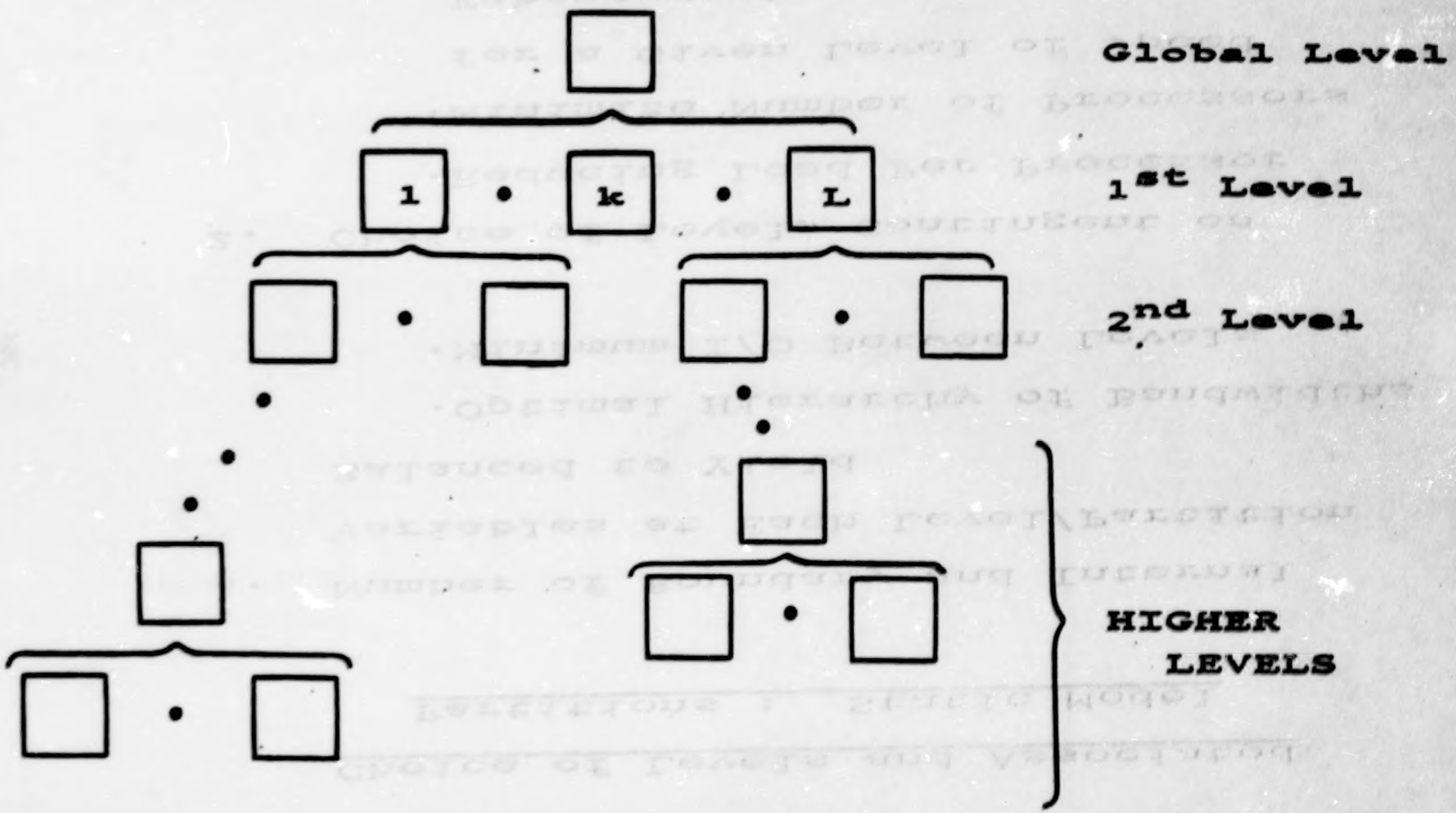
**STEP 1**


GLOBAL
1
k
L
1st Level

**STEP 2**


GLOBAL
1
k
L
1st Level
2nd Level

# DEVELOPMENT OF STATIC POLY TREE


Global Level
1
k
L
1st Level
2nd Level
HIGHER LEVELS

# Choice of Levels and Associated Partitions : Static Model

1. Number of Boundary and Internal Variables at Each Level/Partition Balanced to Yield
   - Optimal Hierarchy of Bandwidths
   - Minimum I/O Between Levels

2. Choice of Levels Contingent on
   - Reducing Load Per Processor
   - Minimize Number of Processors for a Given Level of Speed Enhancement

# DYNAMIC POLY TREE PARALLELISM

## Steps

1. First Level Organized into Convenient Substructural Components (Optimal in Static Sense)

2. Each Substructural Component Refined as Per Local Physics

3. To Maintain Optimality, Refinements May Require Several Levels of Processors

## Choice of Numbers of Level & Associated: Dynamic Model

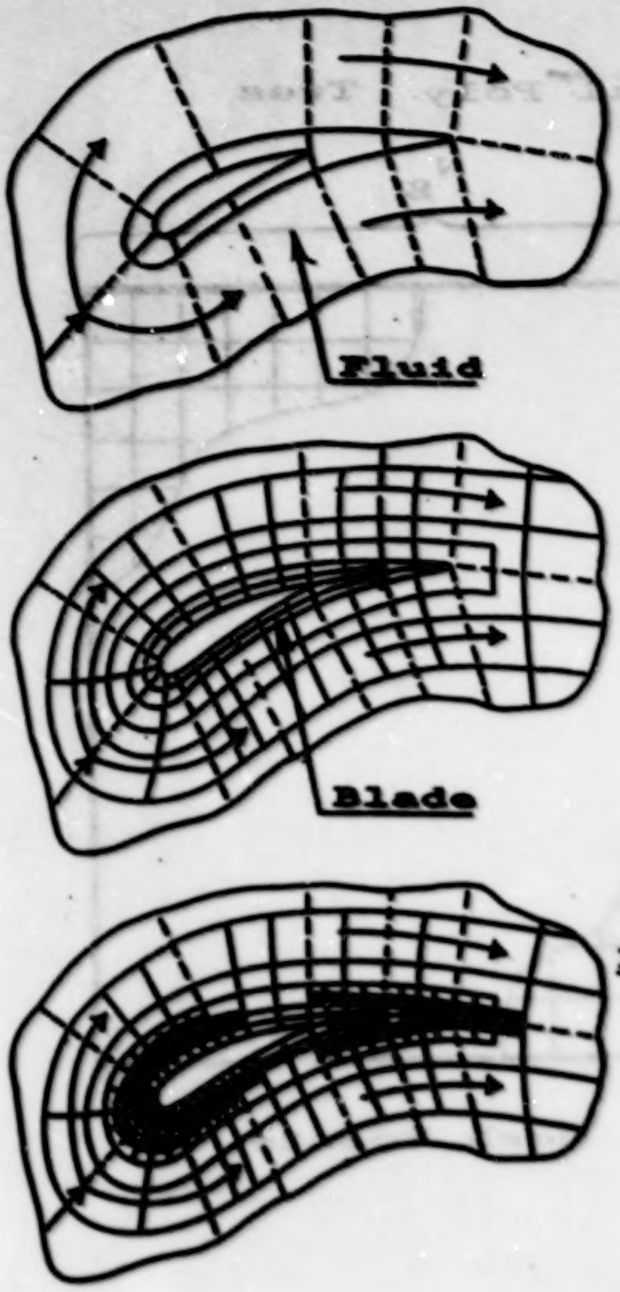
1. First Level Defined as Per Static Tree

2. Choice of Additional Levels and Associated Partitions Contingent on
   - Maintaining Optimality of a Given Branch of Poly Tree
   - Reducing Load Per Processor
   - Minimize Number of Processors
   - Maintain Balance Between Internal and External Variables
   - Minimize I/O Between Levels

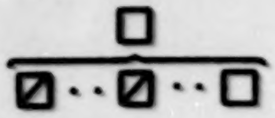
# OPTIMAL PARALLEL COMPUTER ARCHITECTURE FOR INTERDISCIPLINARY MECHANICS SIMULATIONS

**Densifying Solid-Fluid Model**

**Dynamic Poly Tree Arrangement of Processors**



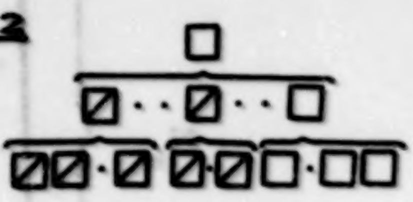
**Level 1**



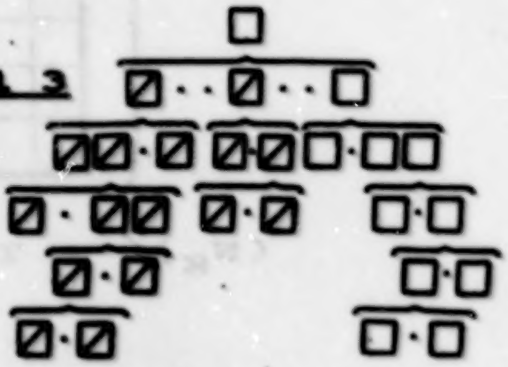
▨ -Fluid Processors

☐ -Solid Processors

**Level 2**



**Level 3**



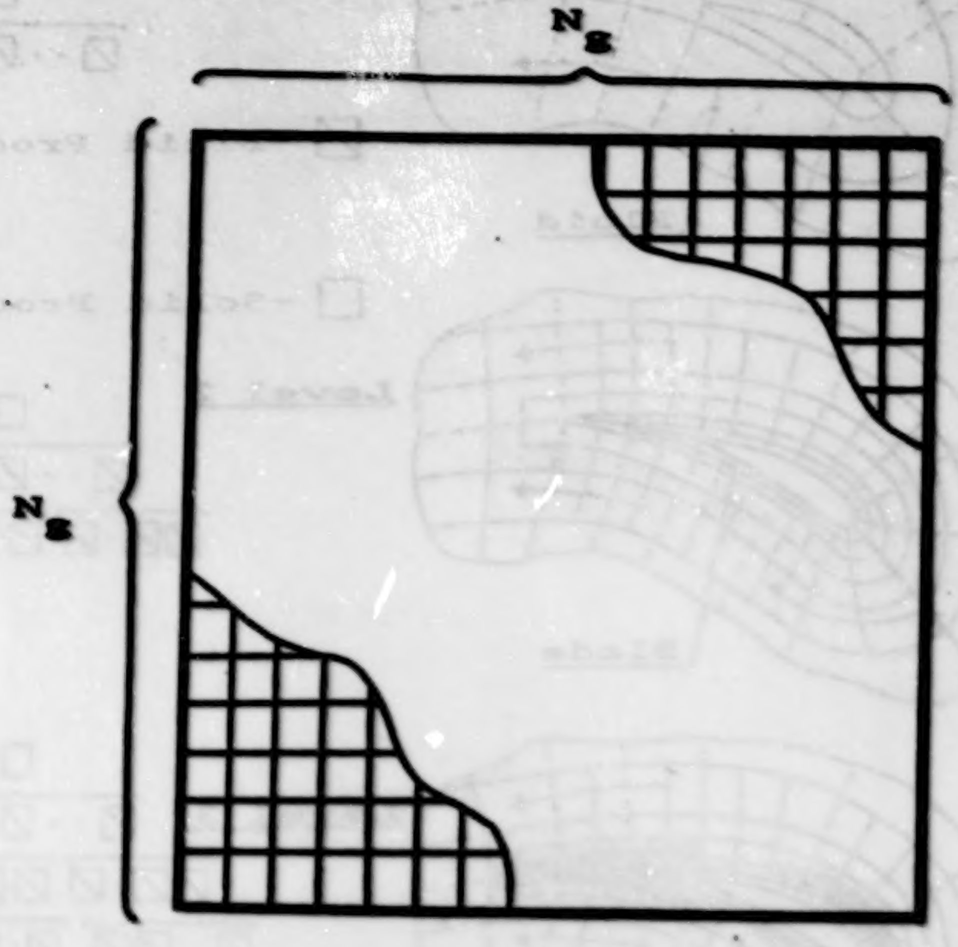
## Example Problem

**Given:**

**Consider Sqare Region With Fine Uniform Mesh**

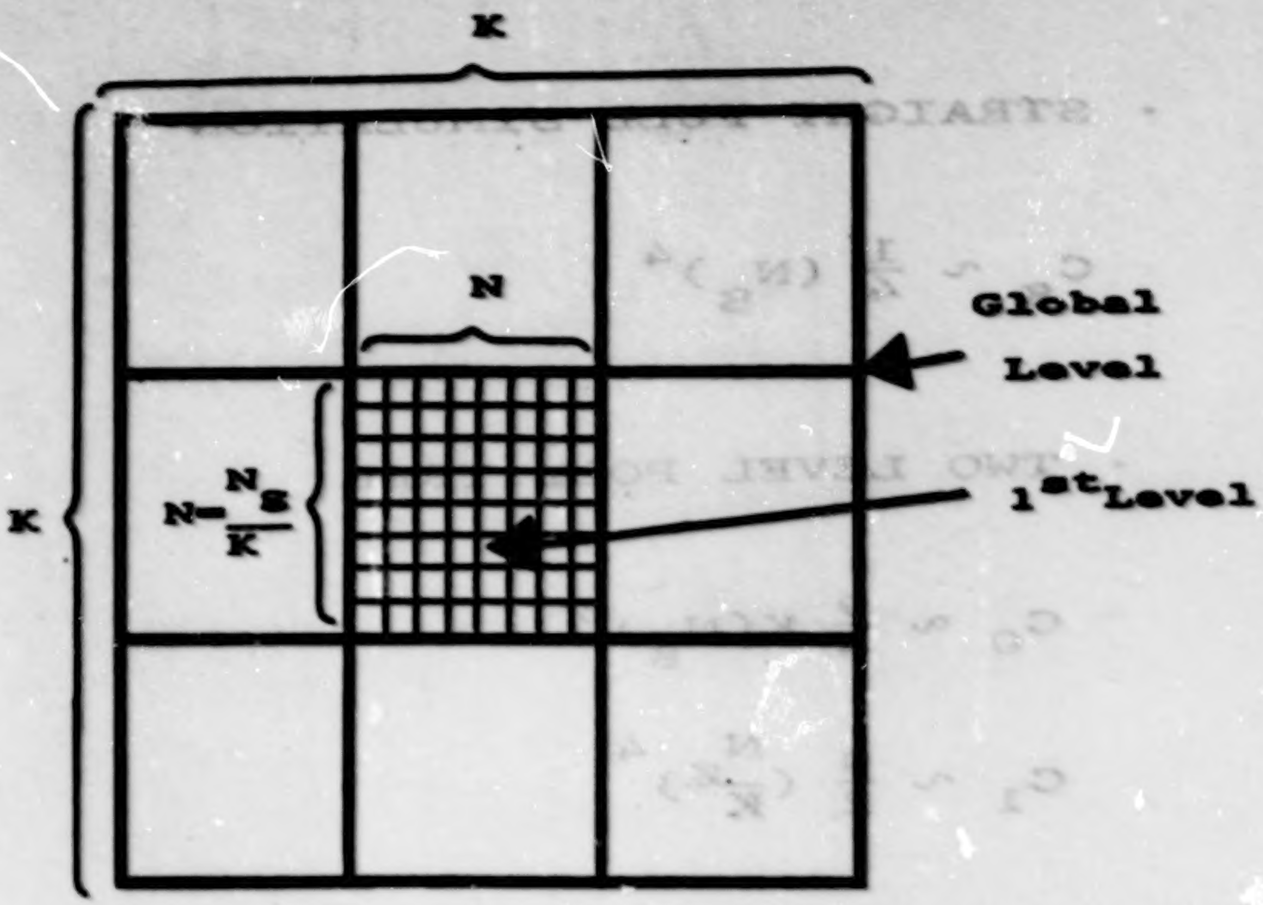
**Problem:**

**Define Optimal Poly Tree**

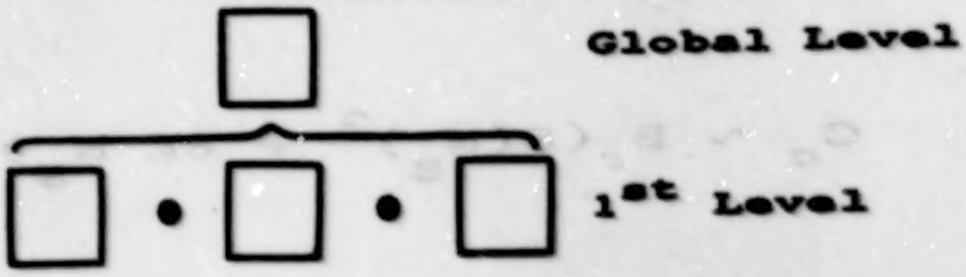


**$(N_g)^2$ - Total Mesh Points**

## TWO LEVEL POLY TREE


K
N
Global
Level
K
N=$\frac{N_s}{K}$
1st Level

## POLY TREE


Global Level
1st Level

$K^2+1$ - Total Number of Processors

# ASYMPTOTIC COMPUTATIONAL EFFORT:

## TWO LEVEL

- STRAIGHT FULL SIMULATION

$$C_g \sim \frac{1}{4}\,(N_g)^4$$

- TWO LEVEL POLY TREE

$$C_0 \sim \frac{9}{4}\,K(N_g)^3$$

$$C_1 \sim \frac{9}{2}\,\left(\frac{N_g}{K}\right)^4$$

- COMMUNICATIONS

$$C_c \sim B_r(8(N_g)^2 + 8K\,N_g)$$

# ASYMPTOTIC COMPUTATIONAL EFFORTS:

## TWO LEVEL

- RATIO COMPARISON

$$R_{p/g} \sim \frac{\psi(C_0+C_1) + \frac{\Omega}{N_c} C_c}{C_g}$$

$$R_{p/g} \sim \psi\left\{\frac{9}{(K)^4} + 4.5 \frac{K}{N_g}\right\} +$$

$$8 \frac{\Omega B_E}{N_c}\left\{\frac{K}{(N_g)^3} + \frac{1}{(N_g)^2}\right\}$$

# OPTIMAL SOLUTION

- GLOBALLY OPTIMIZED

$$\frac{d}{dK} (R_{P/S}) = 0 \rightarrow$$

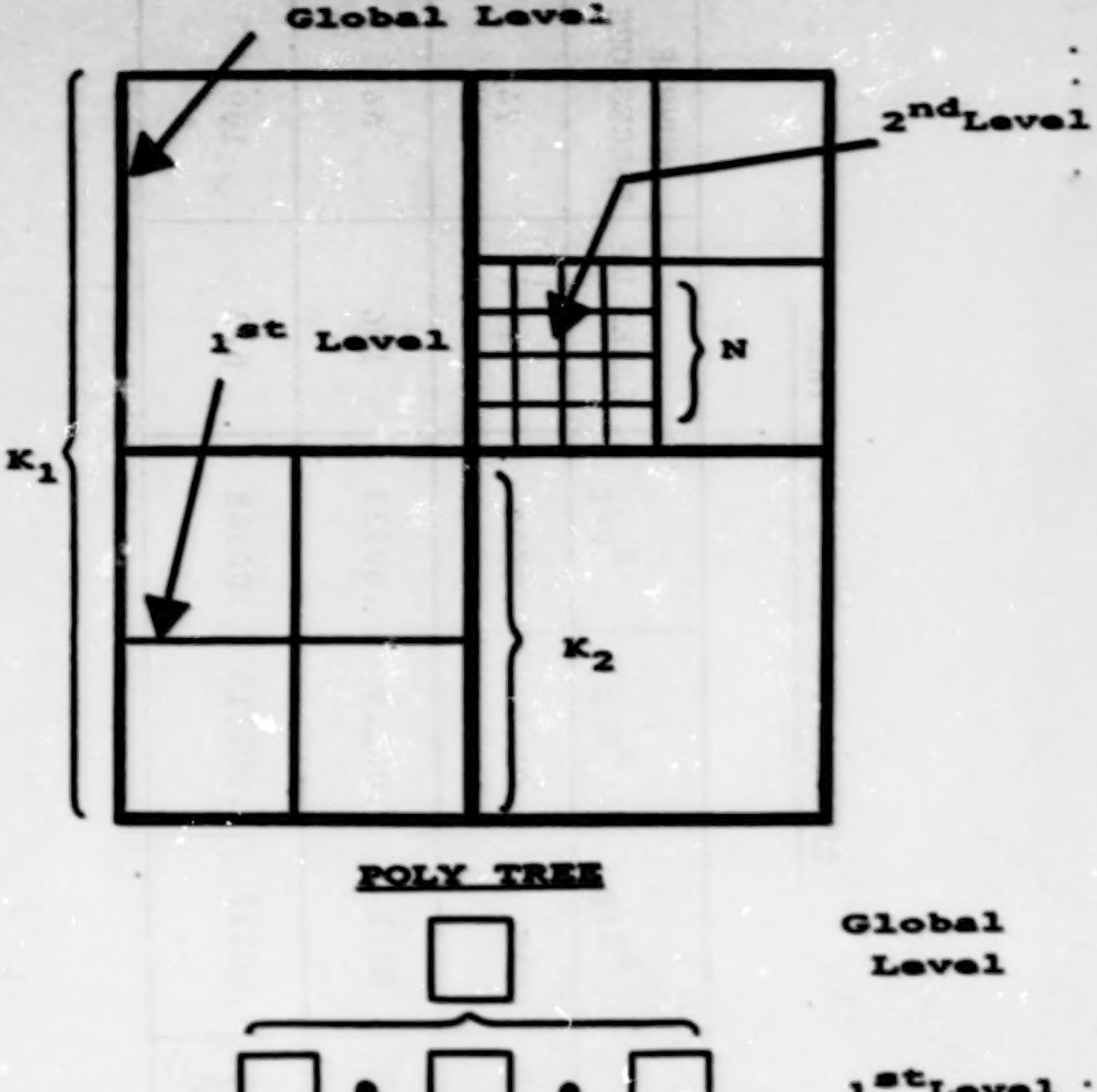
$$K \sim$$

$$\left[ 8\ N_S \left\{ \frac{1}{1 + \frac{16}{9\psi} \frac{\Omega\ B_R}{N_c\ (N_S)^2}} \right\} \right]^{1/5}$$

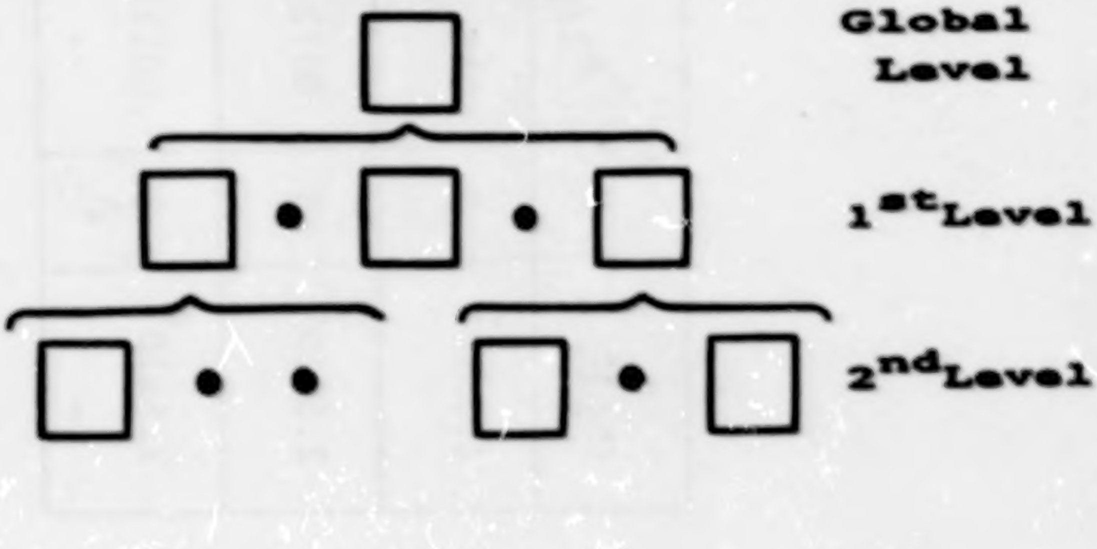
## OPTIMAL TWO LEVEL POLY TREE

| $(N_g)^2$ | K | $R_{1/g}$ | $R_{0/g}$ | $R_{p/g}$ | SPEED UP | NUMBER PROCESSORS |
|---|---|---|---|---|---|---|
| $2.5\times10^5$ | 5 | .045 | .0144 | .0594 | 17 | 24 |
| $2.5\times10^7$ | 8 | .0072 | .00219 | .00939 | 106 | 64 |
| $2.5\times10^9$ | 13 | .00117 | .000315 | .00148 | 673 | 169 |

## THREE LEVEL POLY TREE


Global Level
2nd Level
1st Level
N
$K_1$
$K_2$

### POLY TREE


Global Level
1st Level
2nd Level

## ASYMPTOTIC COMPUTATIONAL EFFORT:

## THREE LEVEL

$$C_0 \sim \frac{9}{4} K_1 (N_g)^3$$

$$C_1 \sim \frac{49}{4} \frac{K_2 (N_g)^3}{(K_1)^3}$$

$$C_2 \sim \frac{9}{4} \left(\frac{N_g}{K_1 K_2}\right)^4$$

$$R_{p/g} \sim$$

$$\underbrace{\frac{9}{(K_1 K_2)^4}}_{\substack{\text{2nd} \\ \text{Level}}} + \underbrace{\frac{49}{2} \frac{K_2}{(K_1)^3 N_g}}_{\substack{\text{1st} \\ \text{Level}}} + \underbrace{4.5 \frac{K_1}{N_g}}_{\substack{\text{0th} \\ \text{Level}}}$$

## SUBOPTIMAL TRENDS: TWO LEVELS

$$R_{p/g} \sim \frac{9}{(K)^4} + 4.5 \frac{K}{N_g}$$

$K \rightarrow$ Large; $K \sim O(N_g)$

$R_{p/g} \sim 4.5$ (450%)

C-5

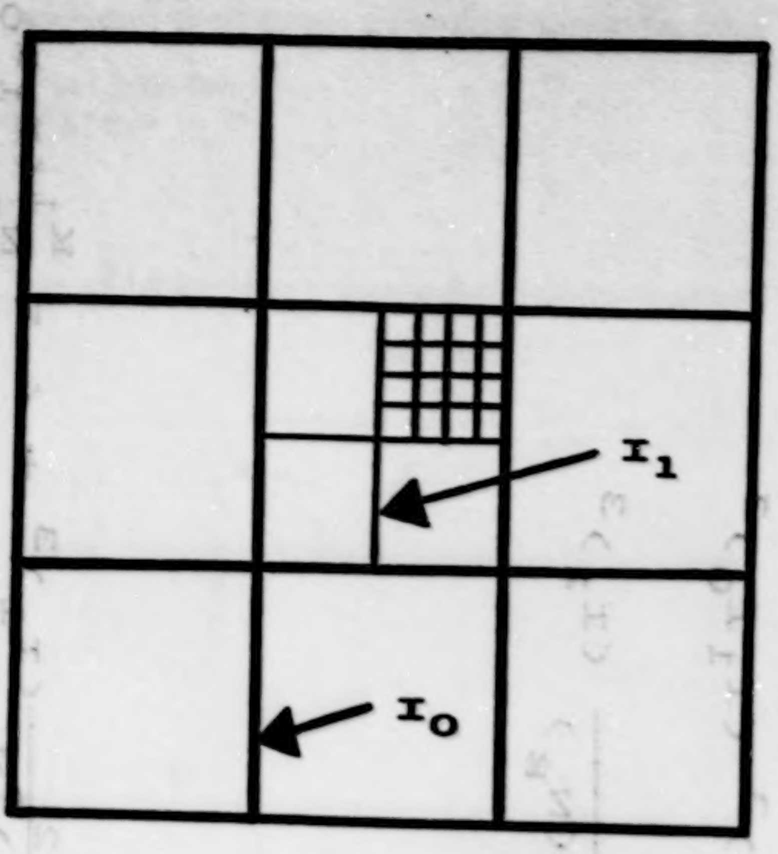
## OPTIMAL THREE LEVEL POLY TREE; $N_g = 5000$

| $K_1/K_2$ | $R_{0/g}$ | $R_{1/g}$ | $R_{2/g}$ | $R_{p/g}$ | SPEED UP | NO. PROCESSORS |
|---|---|---|---|---|---|---|
| 2/6 | $.18\text{x}10^{-2}$ | $.37\text{x}10^{-2}$ | $.4\text{x}10^{-3}$ | $.58\text{x}10^{-2}$ | 170 | 144 |
| 3/5 | $.27\text{x}10^{-2}$ | $.91\text{x}10^{-3}$ | $.18\text{x}10^{-3}$ | $.38\text{x}10^{-2}$ | 264 | 255 |
| 10/4 | $.9\text{x}10^{-2}$ | $.19\text{x}10^{-4}$ | $.35\text{x}10^{-5}$ | $.9\text{x}10^{-2}$ | 110 | 1600 |
|  |  |  |  |  |  |  |

## OPTIMAL THREE LEVEL POLY TREE; $N_g = 50000$

| $K_1/K_2$ | $R_{0/g}$ | $R_{1/g}$ | $R_{2/g}$ | $R_{p/g}$ | SPEED UP | NO. PROCESSORS |
|---|---|---|---|---|---|---|
| 2/8 | $.18\text{x}10^{-3}$ | $.49\text{x}10^{-3}$ | $.13\text{x}10^{-3}$ | $.81\text{x}10^{-3}$ | 1239 | 256 |
| 4/7 | $.36\text{x}10^{-3}$ | $.54\text{x}10^{-4}$ | $.15\text{x}10^{-4}$ | $.43\text{x}10^{-3}$ | 2335 | 784 |
| 10/6 | $.9\text{x}10^{-3}$ | $.29\text{x}10^{-6}$ | $.7\text{x}10^{-6}$ | $.9\text{x}10^{-3}$ | 1110 | 3600 |

# INTERPOLATIVE REDUCTION : 3 LEVEL



| MESH LEVEL | REDUCTION |
|---|---|
| Global | $I_0$ |
| 1st | $I_1$ |
| 2nd | 1 |

## INTERPOLATIVE REDUCTION: 3 LEVEL

$$C_0 \sim \frac{9}{4} K_1 \ (N_g)^3 \ (I_1 I_0)^3$$

$$C_1 \sim \frac{49}{4} \frac{K_2}{(K_1)^3 (N_g)} \ (I_1)^3$$

$$C_2 \sim \frac{9}{(K_1 K_2)^4}$$

$$R_{p/g} \sim$$

$$\underbrace{\frac{9}{(K_1 K_2)^4}}_{\text{2nd Level}} + \underbrace{\frac{49}{2} \frac{K_2}{(K_1)^3 N_g} (I_1)^3}_{\text{1st Level}} + \underbrace{4.5 \frac{K_1}{N_g} (I_1 I_0)^3}_{\text{0th Level}}$$

**REDUCTION EFFECTS: THREE LEVEL POLY TREE;**

$$N_g = 5000$$

$$I_1 = \frac{1}{2} , I_0 = \frac{1}{4}$$

| $K_1/K_2$ | $R_{0/g}$ | $R_{1/g}$ | $R_{2/g}$ | SPEED UP | | NUMBER PROCESSORS |
|---|---|---|---|---|---|---|
| | | | | STRAIGHT | REDUCED | |
| 3/5 | $5.3\text{x}10^{-6}$ | $1.1\text{x}10^{-4}$ | $.18\text{x}10^{-3}$ | 264 | 3386 | 225 |
| 10/4 | $1.7\text{x}10^{-5}$ | $2.3\text{x}10^{-6}$ | $.35\text{x}10^{-5}$ | 110 | 42920 | 1600 |

## REDUCTION EFFECTS: THREE LEVEL POLY TREE;

$N_g = 50000$

$I_1 = \frac{1}{2}$ , $I_0 = \frac{1}{4}$

| $K_1/K_2$ | $R_{0/g}$ | $R_{1/g}$ | $R_{2/g}$ | SPEED UP | | NUMBER PROCESSORS |
|---|---|---|---|---|---|---|
| | | | | STRAIGHT | REDUCED | |
| 4/7 | $7\times10^{-7}$ | $6.7\times10^{-6}$ | $.15\times10^{-4}$ | 2335 | 44,640 | 784 |
| 10/6 | $1.7\times10^{-6}$ | $3.6\times10^{-8}$ | $.7\times10^{-6}$ | 1110 | 402,250 | 3600 |

# Impact on Solvers

Static/Dynamic Poly Tree Architecture Provides a Logical Framework for

- Direct Solvers
- Iterative Solvers
- Mixed (Direct/Iterative) Solvers
- Multi Time Scale Transient Solver
- Local/Global Constrained Nonlinear Solvers
- Mesh Refinement Procedures
- Interpolative Reduction (Saint Venants)
- Etc.

## Summary

The Poly Tree Arrangement Yields

- Optimal Choice of Number of Processors Required for Given Problem
- Reduces Load Per Processor
- Reduces I/O Between Processors
- Enables Optimal Handling of Automatic Mesh Refinement
- Provide Most Natural Route for I/O Flow
- Enables an Orderly Way to Perform Interpolative Mesh Refinement

## Future Directions

1. Continue Refinement of Scheme
2. Develop Associated Parallel Solution Procedure
    - Direct
    - Iterative
    - Mixed
    - Steady State
    - Transient
3. Establish requirements of Data Based Management System Required for Overall Contol

☆ U.S. GOVERNMENT PRINTING OFFICE:

National Aeronautics and Space Administration

# Report Documentation Page

1. Report No.
NASA CP-10012, Part 1

2. Government Accession No.

3. Recipient's Catalog No.

4. Title and Subtitle
NASA Workshop on Computational Structural Mechanics - 1987

5. Report Date
February 1989

6. Performing Organization Code

7. Author(s)
Nancy P. Sykes, Editor

8. Performing Organization Report No.

10. Work Unit No.
505-63-01-10

9. Performing Organization Name and Address
NASA Langley Research Center
Hampton, VA 23665-5225

11. Contract or Grant No.

13. Type of Report and Period Covered
Conference Publication

12. Sponsoring Agency Name and Address
National Aeronautics and Space Administration
Washington, DC 20546-0001

14. Sponsoring Agency Code

15. Supplementary Notes
Nancy P. Sykes: Analytical Services and Materials, Inc., Hampton, Virginia.

16. Abstract

This conference publication contains the proceedings of the Workshop on Computational Structural Mechanics held at NASA Langley Research Center, November 18-20, 1987. The workshop was sponsored jointly by NASA Langley Research Center and NASA Lewis Research Center.

The workshop was organized into the following three sessions:

(1) Concurrent Processing Methods and Applications
(2) Advanced Methods & Testbed/Simulator Development
(3) Computational Dynamics

17. Key Words (Suggested by Author(s))
Parallel Processing
Computational Dynamics
Testbed
Computational Structural Mechanics

18. Distribution Statement
Unclassified - Unlimited
Subject Category 39

19. Security Classif. (of this report)
Unclassified

20. Security Classif. (of this page)
Unclassified

21. No. of pages
392

22. Price
A17

NASA FORM 1626 OCT 86

For sale by the National Technical Information Service, Springfield, Virginia 22161-2171

END

DATE

FILMED

DEC 5 1989